HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Peter Hofmann

Entwicklung und Validierung einer prototypischen Lösung zur gestenbasierten Steuerung von PowerPoint Präsentation mithilfe von Kinect

Mittweida, 2016

Fakultät Medien

DIPLOMARBEIT

Entwicklung und Validierung einer prototypischen Lösung zur gestenbasierten Steuerung von PowerPoint Präsentation mithilfe von Kinect

Autor:
Peter Hofmann

Studiengang:
Multimediatechnik

Seminargruppe:
MK11w1-D

Erstprüfer:
Prof. Dr.-Ing. Frank Zimmer

Zweitprüfer:
Dipl.-Ing. Norbert Göbel

Einreichung:
Mittweida, 15.01.2016

Verteidigung/Bewertung:
Mittweida, 2016

Bibliografische Beschreibung:

Peter Hofmann:
Entwicklung und Validierung einer prototypischen Lösung zur gestenbasierten Steuerung von PowerPoint Präsentation mithilfe von Kinect. -2016. -XI. -71, -10 S.
Mittweida, Hochschule Mittweida, Fakultät Medien, Diplomarbeit 2016

Referat:

Die vorliegende Arbeit befasst sich mit der Entwicklung einer prototypischen Lösung zur Steuerung von PowerPoint Präsentationen mithilfe von KINECT. Dabei wird der Vorgang vom Entwurf bis hin zur Implementierung beschrieben. Neben der Entwicklung wird sich auch mit der Validierung dieser Lösung befasst. Dabei soll die Gestenerkennung der Microsoft KINECT v2 überprüft werden. In die Grundlagen dieser Arbeit fließen der Aufbau und die Funktionsweise der Microsoft KINECT v2 mit ein.

Inhalt

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abb	Abbildung
API	Application Programming Interface
CGI	Computer Generated Imagery
DLL	Dynamic Link Library
SDK	Software Development Kit
TOF	Time of Flight
VGB	Visual Gesture Builder
WPF	Windows Presentation Foundation

1 Einleitung

Im einleitenden Kapitel werden die Motivation und die Aufgabenstellung dieser Diplomarbeit besprochen. Gleichzeitig erfolgt ein kurzer Überblick zu den einzelnen Kapiteln dieser Arbeit.

1.1 Motivation

Die berührungslose Interaktion mit dem Computer ist kein Traum mehr. Wie aus Filmen wie „Minority Report“, als die Darsteller größtenteils die Computerinteraktion über Gesten steuerten oder „Iron Man“ aus dem Jahr 2008, als Robert Downey Jr. aka Tony Stark mit seinem Computerprogramm „JARVIS“ nur mittels Handbewegung Hologramme erstellt, verschiebt oder bearbeitet, bekannt, werden langsam Realität.

Seit Jahren zeigen uns Science-Fiction Filme einfallsreiche Arten zur Steuerung virtueller Objekte. Im Film geschieht dies natürlich nur mittels CGI Effekte, aber die Faszination an den berührungslosen Interaktionen ist daher enorm und bietet uns Ansporn zur Verwirklichung.

IT-Systeme müssen heute nicht mehr immer klassisch mit Maus und Tastatur bedient werden. In den letzten Jahren ergaben sich eine Reihe von Alternativen für die Bedienung von Geräten und Softwaresystemen. Neben der Steuerung durch einen entsprechenden Datenhandschuh oder Spracherkennungssoftware, ist das wohl bekannteste und weitverbreitetste Beispiel für eine alternative Steuerung: Touchscreens. Sie ermöglichen es Geräte, wie Smartphone und Tablets mittels Gesten zu steuern. So kann per „Wischgeste“ zwischen verschiedenen Bildschirmen gewechselt werden oder mit zwei diagonal angesetzten Fingerspitzen können Fenster vergrößert oder verkleinert werden. Der neueste Trend sind berührungslose Interaktionen durch Gesten. Vor einiger Zeit noch sehr kostenintensiv, kommt ausgerechnet aus der Videospielbranche mit der Microsoft KINECT eine kostengünstige Alternative.

Aktuell besteht noch der Eindruck, dass die meisten berührungslosen und gestengesteuerten Anwendungen Spiele sind. Durchaus bietet sich diese Technik besonders bei Spielen an, da auch aufgrund der nicht ganz hundertprozentigen Genauigkeit der Technik es bei Fehlern zu keinen gravierenden Auswirkungen kommen kann. Jedoch ist der Einsatz von Gestensteuerung in anderen Bereich durchaus denkbar.

Die Microsoft KINECT wurde ebenfalls für die Videospielbranche entwickelt, allerdings ist sie mit dem Computer kompatibel. Dank ihrer Kompatibilität zum Computer und ihrer neuartigen Technik verhilft uns die KINECT zu einer neuen Möglichkeit der berührungslosen Interaktion mit IT-Systemen in allen Bereichen.

1.2 Zielsetzung

Das Hauptziel dieser Arbeit, ist die Entwicklung einer prototypischen Lösung zur gestenbasierten Steuerung von PowerPoint-Präsentationen unter Verwendung der Microsoft KINECT v2. Es soll möglich sein, allein mithilfe von Handbewegungen durch eine PowerPoint-Präsentation zu leiten.

Im Rahmen der Aufgabenstellung befasst sich diese Arbeit mit der von Microsoft entwickelten KINECT v2 Kamera. Dafür wird der Aufbau, die Funktionsweise und Unterschied zur ersten KINECT beleuchtet.

Die Entwicklung der prototypischen Anwendung stellt den Schwerpunkt und Kern der Arbeit dar. Die Arbeit beschäftigt sich zusätzlich mit der Fehleranalyse der Microsoft KINECT. Anhand der prototypischen Anwendung werden mittels unterschiedlicher Personen Tests durchgeführt. Für ein ausschlaggebendes Ergebnis und um ein möglichst großes Spektrum an Nutzern abzudecken, werden die Testpersonen sich in gezielten Kriterien unterscheiden. Falls es während der Tests zu Problemen kommt, soll so ermittelt werden, ob deren Auftreten auf qualitative Mängel der KINECT-Datenerfassung zurückzuführen ist oder durch ungenaue Nutzerinteraktionen. Mithilfe dieser Erkenntnisse wird anschließend geprüft, wie man diese Probleme beheben bzw. mindern kann.

1.3 Kapitelübersicht

Diese Diplomarbeit unterteilt sich in sieben Kapitel. Nach dem **Kapitel 1**, der Einleitung, welche die Motivation und die Zielsetzung zu dieser Arbeit beinhaltet, wird sich das **Kapitel 2** ausschließlich mit der Microsoft KINECT befassen. Es soll deutlich gemacht werden, was Microsoft KINECT überhaupt ist, woher die KINECT kommt und was das eigentliche Ziel der KINECT Kamera war. Im Weiteren wird der KINECT - Sensor im Detail vorgestellt, wie ist der Sensor aufgebaut und wie die einzelnen Bestandteile der KINECT arbeiten. Da die Entwicklung der prototypischen Anwendung mit der Microsoft KINECT der zweiten Generation erfolgt, befasst sich das Kapitel abschließend noch mit den Unterschieden der beiden KINECT Versionen.

Das **Kapitel 3** beschäftigt sich mit den Grundlagen und Randbedingungen, welche für die Entwicklung einer KINECT Anwendung notwendig sind. Die Umsetzung der KINECT - Anwendung erfolgt mittels der Programmiersprache C# und unter Hilfe des von Microsoft für KINECT entwickelte „Windows Developer Toolkit“.

Die **Kapitel 4** und **5** beziehen sich auf die eigentliche Realisierung der Anwendung. Zunächst wird im **Kapitel 4** noch der Entwurf spezifiziert, ehe im nächsten Kapitel die Umsetzung thematisiert wird. Es wird auf die Realisierung der KINECT - Anwendung eingegangen, sowie die Umsetzung der Gesten zur Steuerung der Präsentationen.

Im **Kapitel 6** wird die Thematik „Validierung“ angestrebt. Zunächst wird erläutert, wie die Vorgehensweise zur Überprüfung der Funktionalität ist. Anschließend werden die Kriterien der Testpersonen vorgestellt. Abgeschlossen wird das Kapitel mit der Darstellung der Testdurchführung.

Schließlich werden im **Kapitel 7** die Ergebnisse aus dem vorherigen Kapitel zusammengefasst und die Resultate zur Genauigkeit und Fehlertoleranz ausgewertet. Die daraus gewonnenen Erkenntnisse werden anschließend präsentiert. Zum Schluss wird ein Ausblick wiedergegeben, inwieweit die Microsoft KINECT für Realisierung der Thematik dieser Arbeit geeignet ist.

2 Microsoft KINECT

Der Begriff „KINECT“, „KINECT Kamera“ oder „KINECT Sensor“ ist noch nicht weitverbreitet. Hin und wieder hört man ihn im Zusammenhang mit der Spielekonsole Xbox 360 oder Xbox One. Leider können viele mit diesem Begriff nichts anfangen, obwohl der Begriff schon viel verrät, wenn man seine Herkunft kennt.

Aus der Einleitung dieser Arbeit kann man entnehmen, dass die Microsoft KINECT die Möglichkeit bietet, auf Gesten zu reagieren. Die KINECT bietet aber wesentlich mehr als nur diese Funktion. Um diese Arbeit besser verstehen zu können, wird in den nachfolgenden Punkten der Begriff „KINECT“ erläutert. Es wird geklärt, was KINECT ist und wie sie funktioniert.

2.1 Was ist Microsoft KINECT?

Der Begriff KINECT ist von dem englischen Begriff „kinetic connect“ abgeleitet, was übersetzt so viel wie „kinetische Verbindung“ bedeutet. Dies erklärt schon ziemlich gut, was KINECT eigentlich ist, nämlich eine Verbindung die auf Bewegung reagiert.

Der KINECT Sensor wurde unter „Project Natal“ von Microsoft und PrimeSense entwickelt. PrimeSense ist eine israelische Firma, die 2005 gegründet wurde. Sie beschäftigte sich unteranderem mit der Entwicklung von 3D-Sensoren. 2013 wurde PrimeSense von Apple Inc. aufgekauft und arbeitet seitdem in deren Auftrag.

KINECT wurde erstmals 2009 auf der Spielemesse E3, noch unter seinem Projektnamen, vorgestellt. Ihr eigentliches Einsatzgebiet war in der Videospielbranche.

Um neue Spieler und eine größere Zielgruppe zu erreichen, setzten die großen Spielkonsolen Hersteller Sony, Nintendo und Microsoft auf eine neue Art der Interaktion, die den herkömmlichen Controller ablösen sollte. Der erste seiner Art war die EyeToy Kamera (Abb. 2-1) von Sony.

Abbildung 2-1: EyeToy Kamera

EyeToy erschien 2003 für die damalige Playstation 2. Sie nutzte für die Gestensteuerung den Ansatz der Stereoskopie, es ließen sich Bewegungen des Spielers, durch die Kamera aufzeichnen und in das Spielgeschehen einbeziehen.

Während EyeToy nur ein zusätzliches Gadget zur Playstation 2 war, setzte Nintendo bei seiner neuen Konsolengeneration 2006, der Nintendo Wii, komplett auf eine Bewegungssteuerung. Anders als Sony setzte man aber nicht auf eine vollständig freie Interaktion mit der Konsole, sondern man entwickelte die sogenannte Wii Remote Fernbedienung (Abb. 2-2). Diese Technik ermöglichte es, über Sensoren im inneren der Fernbedienung die Lage und die Bewegungen des Spielers zu erfassen und an die Konsole zusenden. Hinzu kommen Infrarot-Informationen, welche von einer unterhalb bzw. oberhalb des Fernsehers angebrachten Sensorleiste aufgenommen werden, in etwa wie bei einem Laserpointer. Diese Kombination ermöglicht es dem Nutzer das Spiel durch körperliche Gesten sowie durch Tasten zu steuern.

Abbildung 2-2: Wii Remote Fernbedienung + Sensorleiste

Das Wagnis von Nintendo, komplett auf eine neue Art der Interaktion zusetzen, zahlte sich aus. In den Jahren 2007 bis 2010 war die Nintendo Wii, die am häufigsten verkaufte Konsole in Deutschland [WII_2-1]. Der Vorteil war, dass man mit der neuen Steuerungsart eine neue Zielgruppe erreichte und so viele neue Spieler gewinnen konnte. Sony, aber vor allem Microsoft, die bis dahin gar keine neue Art der Interaktion auf dem Markt hatten, mussten handeln, um nicht den Anschluss an Nintendo zu verlieren.

Sony setzte daraufhin 2010 mit Playstation Move (Abb. 2-3) auf eine ähnliche Technik wie Nintendo. Mittels eines Motion Controllers werden hier die Gesten des Spielers über eine Kamera erfasst und ins Spiel transportiert.

Abbildung 2-3: Playstation Move Motion Controller + Kamera

Microsoft hingegen setzte auf eine vollständig neue Art der Interaktion. Mit einer Spielumgebung ganz ohne irgendeine Art eines Controllers revolutionierte Microsoft das, was von Sony und Nintendo einst begonnen wurde.

Komplett ohne Controller, allein die KINECT Kamera (Abb. 2-4) analysiert während des Spiels die Bewegung des Spielers. Dabei geht die Analyse der Bewegung weit über die der EyeToy Kamera hinaus. Anders als die EyeToy Kamera erfasst die KINECT den Körper dreidimensional. Mittels verschiedener Kameras wird ein 3D-Abbild eines Körpers erstellt. Anschließend werden die Positionen spezieller Körperteile, wie z.B. Kopf, Schultern oder Knie analysiert und direkt in das Spiel eingebunden. So lassen sich unter anderem Bewegungen wie Sprünge und Drehungen ins Spiel übertragen.

Abbildung 2-4: KINECT

Das Potenzial der KINECT wurde schnell erkannt und über Einsatzmöglichkeiten außerhalb der Spielebranche nachgedacht. Microsoft veröffentlichte 2011 eine KINECT für PC Anwender. Unter dem Name „KINECT for Windows“ wurde ein KINECT - Sensor auf dem Markt gebracht, der für Entwickler die Möglichkeit bot, eigene Anwendungen zu programmieren.

Für die Programmierung stellt Microsoft im selben Jahr noch ein kostenloses Software Development Kit (SDK) bereit. Neben dem kompatiblen Treiber für den KINECT Sensor bot das SDK die Möglichkeit, mit den Programmiersprachen C++, C# oder Visual Basic eigene Anwendungen zu entwickeln. Zunächst nur für Windows 7 erhältlich, ist das SDK mittlerweile auch für Windows 8 und 8.1 sowie Windows 10 verfügbar. Ausführliche Informationen zum SDK folgen im nächsten Kapitel.

Soviel Innovatives in der KINECT auch steckt und so viel Einsatzmöglichkeiten sich, durch die PC Unterstützung auch aufbot, die KINECT hatte ihre Grenzen. Vor allem die geringe Auflösung und das beschränkte Sichtfeld machten der KINECT zu schaffen. Bei einem größeren Abstand zur Kamera, sind aufgrund der geringen Auflösung einige Details nicht mehr zu erkennen, andererseits sind bei einem geringen Abstand zwar viele Details zuerkennen, aber wegen dem begrenzten Sichtfeld sind das Objekt oder die Person nicht mehr vollständig im Bild. Hinzukommen Verzögerungen bei der Verarbeitung. Von zu hohen Latenzzeiten ist oft die Rede. Darüber hinaus ist die KINECT sehr anfällig gegen Fremdlicht. Durch einfallendes Sonnenlicht konnte es daher zu Fehlmessungen kommen [KIN_2-2].

Mit Erscheinen der neuen Konsolen Generation brachte Microsoft im November 2013 eine überarbeitete KINECT Kamera (Abb. 2-5) auf den Markt.

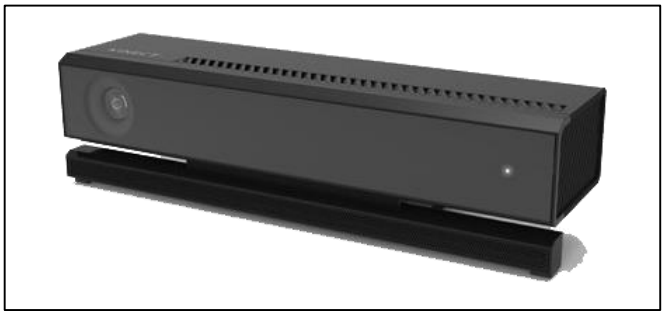


Abbildung 2-5: KINECT for Windows v2

Unter dem Namen „KINECT v2“ oder „KINECT for Xbox ONE“ war diese nicht nur äußerlich neu, sondern brachte auch technisch einige Änderungen mit. Zunächst nur zusammen mit der Xbox ONE erhältlich, erschien erst Mitte 2014 die neue KINECT für den PC. Man hofft nun bei Microsoft mit der KINECT v2 alle Kritikpunkte der ersten KINECT aufgearbeitet und ein Gerät auf dem Markt zu haben, das in allen Bereichen Zuspruch erhält.

2.2 Aufbau der Microsoft KINECT

Die KINECT der zweiten Generation besteht, wie bereits ihr Vorgänger, aus mehreren einzelnen Komponenten, wobei vier Komponenten (Abb. 2-6) für die Funktionalität der KINECT rausstechen. Dazu gehören die Farb- und Tiefenkamera, der Infrarotprojektor sowie die Mikrofone.

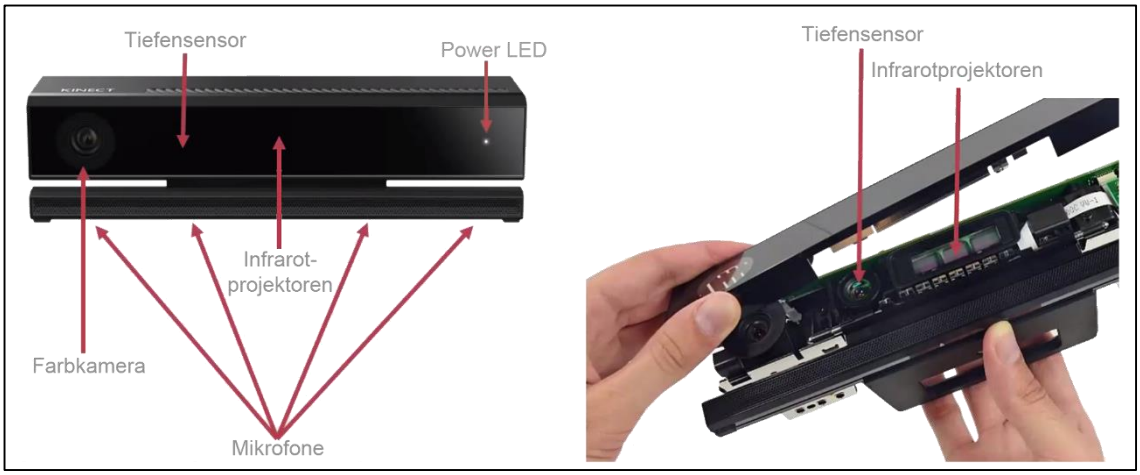


Abbildung 2-6: Aufbau KINECT v2

Im Vergleich zur vorhergingen Version, wurden alle Komponenten deutlich verbessert, was sich vor allem an dem vergrößerten Sichtfeld der KINECT verdeutlicht. Hatte die alte Version noch ein Sichtradius von vertikal 43° und horizontal 57° (wobei die 43°nur durch

den integrierten Neigungsmotor zu erreichen waren), so bietet die neue KINECT ohne Unterstützung eines Neigungsmotors einen Sichtradius von vertikal 60° und horizontal 70°.

Die KINECT v2 verwendet nun eine High-Definition (HD) fähige Farbkamera, welche Bilder im Vollbildverfahren mit vertikal 1080 Pixeln (1080p) erfassen kann. Dies ermöglicht Bilder und Videos in einer Auflösung von 1920 x 1080 Pixeln. In Abhängigkeit der Lichtbedingungen arbeitet die Kamera mit einer Bildfrequenz von bis zu 30 Bildern pro Sekunde. Bei niedrigen Lichtverhältnissen wird die Bildfrequenz der Kamera allerdings heruntergesetzt und die Aufnahmen werden mit 15 Bildern pro Sekunde erfasst.

Die Tiefenkamera wurde im Vergleich zu vorherigen Version um das Dreifache verbessert. Sie kann nun Bilder mit einer Auflösung von 512 x 424 Pixeln erfassen. Die Standardreichweite der KINECT v2 wird von Microsoft mit 0,5 bis 4,5 Metern angegeben. Die Tiefenkamera ist jedoch in der Lage, Personen und Gegenstände bis zu einer Reichweite von acht Metern war zunehmen [KIN_2-3]. Dabei ist in diesem Bereich aber keine Skelettverfolgung mehr möglich.

Die erste KINECT Generation funktionierte nach dem „Structured-Light“ Prinzip. Der Infrarotprojektor der KINECT v2 arbeitet nun nach dem Prinzip der „Time-of-Flight“. Im Gegensatz zum „Structured-Light“ Prinzip, was weiter entfernte Objekte unsauber erfasst, wird durch das „Time-of-Flight“ Prinzip eine höhere Genauigkeit erzielt. Hinzu kommt, dass der Infrarotprojektor nun wesentlich unempfindlicher gegen Beleuchtung im Raum ist.

Wie auch die erste Version besitzt auch die neue Version vier einzelne Mikrofone, welche zusammen in einer Leiste unterhalb der Kameras angebracht sind. Jedes einzelne Mikrofon kann Töne bereits ab 20dB war nehmen. Außerdem sind diese nun in der Lage, nicht nur anhand der Lautstärke die Entfernung zu der jeweiligen Person, sondern auch die exakte Position zu ermitteln. Sie erkennen, ob die Person links, rechts oder frontal vor der KINECT spricht, und können so die Position zuweisen. Dabei nehmen die Mikrofone Töne in einen Radius von + und -50° auf.

2.3 Funktionsweise der Microsoft KINECT

Nach dem der Aufbau des KINECT v2 Sensors geklärt ist, stellt sich natürlich die Frage: Wie funktioniert KINECT? Hierfür ist das Prinzip des „Time-of-Flight“ wichtig.

„Time-of-Flight“ (TOF), übersetzt „Flugzeit“, basiert auf der Berechnung von Distanzen durch das Licht. Grob gesagt, misst das TOF-Prinzip, wie lange Photonen bis zu einem Objekt und wieder zurück benötigen. Man kann sich das ähnlich der Echoortung bei Fledermäusen vorstellen. Hier werden Ultraschallwellen von der Fledermaus ausgestoßen, welche von Objekten reflektiert werden. Diese nimmt die Fledermaus wieder auf und das

Gehirn der Fledermaus kann durch die Zeitunterschiede der zurückkehrenden Schallwellen ein Abbild der Umgebung erstellen. So orten Fledermäuse, wie weit ein Baum oder ein Insekt entfernt ist.

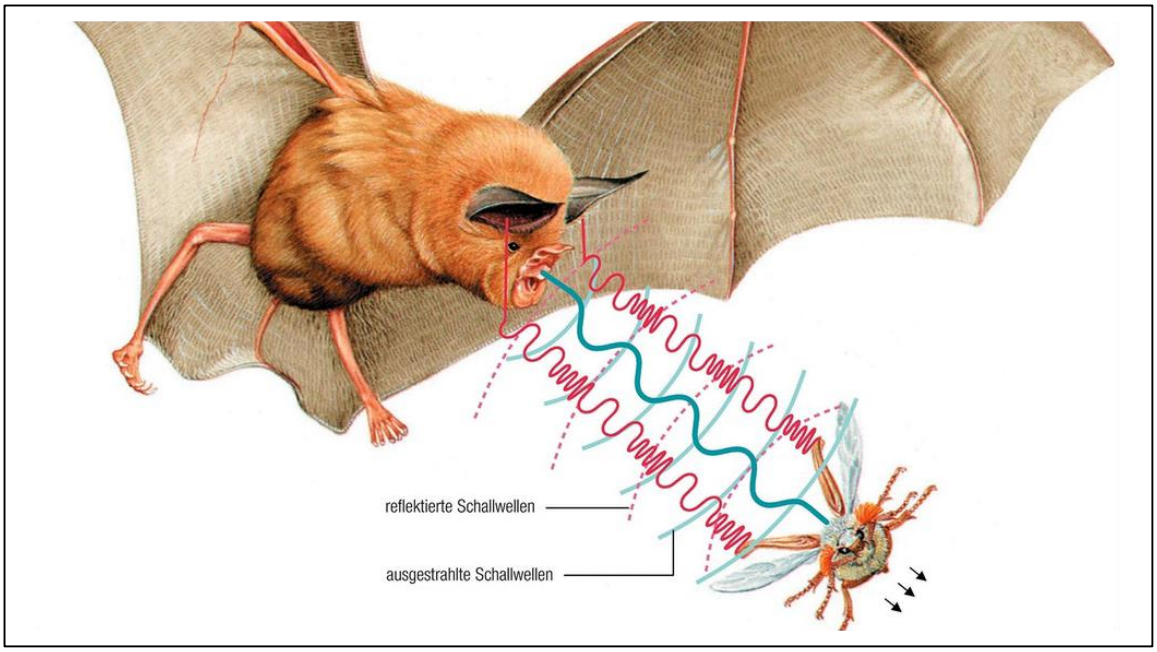


Abbildung 2-7: Echoortung Fledermaus

Bei der KINECT sind es natürlich keine Ultraschallwellen, sondern ein, von den drei Infrarotprojektoren, großflächiger ausgesendeter Infrarotimpuls. Dieser wird nun von den Personen oder Gegenstände vor der KINECT als Reflexion zurückgeworfen. Die KINECT nimmt diese Reflexionen auf und misst anschließend für jeden einzelnen Pixel, wie lange das Licht unterwegs war, bevor es zurück zum Sensor reflektiert wurde. Das heißt, es werden aufgrund der neuen Auflösung der Tiefenkamera 512 x 484 Pixel gemessen. Dadurch wird ein 3D-Abbild der Szene konstruiert. Durch die verbesserte Optik lassen sich Objekte ab einer Distanz von 0,5 bis 4,5 Metern vollständig erfassen.

2.4 Skelettverfolgung

Alleine mit den Farb- und Tiefendaten aus den jeweiligen Kameras oder der Spracherkennung ließen sich interessante Anwendungen mit der KINECT entwickeln. Zur Realisierung des Hauptziels dieser Arbeit dient aber wohl das charakteristischste Feature der KINECT, die Bewegungsverfolgung des Spielers, die sogenannte Skelettverfolgung (Skeleton Tracking).

Bei der Skelettverfolgung wird vom KINECT Sensor ermittelt, wo sich die Körperteile des Spielers befinden. Damit wird festgestellt, wie sich der Spieler vor dem Sensor bewegt. Dies geschieht, durch hinterlegte Informationen über die Gelenkpunkte (Abb. 2-8) des menschlichen Skeletts in einer Datenbank. Die erfassten Farb- und Tiefendaten werden mithilfe der Datenbank in Koordinaten von Skelettteilen umgewandelt. Das heißt, wird

eine Person erkannt, so liefert die KINECT die räumlichen Koordinaten der jeweiligen Körperteile. So kann der Abstand von z. B. Kopf und Hand berechnet werden.

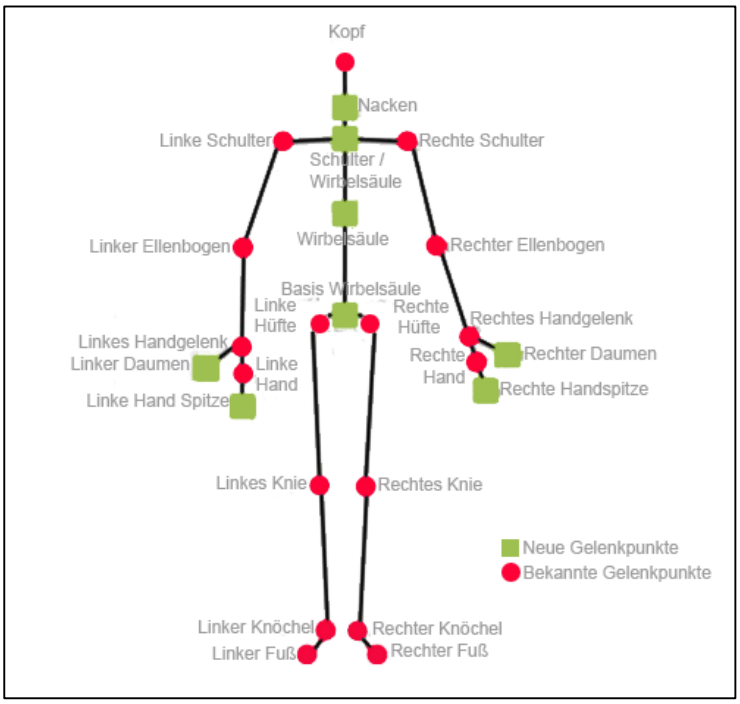


Abbildung 2-8: Gelenkpunkte KINECT v2

Die KINECT v2 ist in der Lage, sechs Personen gleichzeitig wahrzunehmen und jede wahrgenommene Person mit je 25 Gelenkpunkte zu analysieren. Im Gegensatz konnte die erste Generation der KINECT zwar ebenfalls bis zu sechs Personen wahrnehmen, aber nur zwei davon mit je 20 Gelenkpunkte erfassen.

Insbesondere im Gebiet der Hüfte, der Wirbelsäule und der Schulter wurden neue Gelenkpunkte hinzugefügt, dies bringt eine bessere Analyse im Bereich Neigung und Rotation im Oberkörper. Wie die „KINECT for Windows", bietet auch die neue KINECT einen „Seated Mode“. In diesen Modus erkennt die KINECT ebenfalls alle Gelenkpunkte, es werden lediglich die 10 Gelenkpunkte aus dem unteren Körperbereich ignoriert.

Auch die Hände wurden im Vergleich zur ersten KINECT verbessert. Gab es bei der ersten KINECT nur rechtes und linkes Handgelenk und rechte und linke Hand, so gibt es nun zusätzlich noch rechte und linke Handspitze und rechter und linker Daumen, damit lässt sich nun auch der Status der Hand festlegen. Die drei Standardpositionen der Hand sind „Offen“, „Geschlossen“ und „Lasso“ (Abb. 2-9).

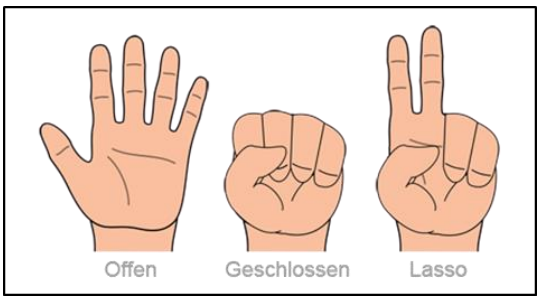


Abbildung 2-9: Standard Handzustände der KINECT v2

Die Reichweite der Skelettverfolgung entspricht die der kompletten Reichweite der KINECT v2. So lässt sich die Skelettverfolgung von 0,5 bis 4,5 Metern erkennen. Jedoch empfiehlt Microsoft bei der Skelettverfolgung den Bereich 0,8 bis 3,5 Metern. [KIN_2-4]

2.5 Unterschied zwischen KINECT und KINECT v2

Die folgende Tabelle zeigt die beiden KINECT Version im Vergleich.

	KINECT 1. Generation	KINECT 2. Generation
Veröffentlichung	Für Xbox 360: 10. November 2010 Für PC: 21. Februar 2011	Für Xbox One: 22. November 2013 Für PC: 15. Juli 2014
Sichtfeld	Horizontal: 57° Vertikal: 43°	Horizontal: 70° Vertikal: 60°
Reichweite	max.: ~ 4m min.: 0,8m im Near Mode: 0,4m	max.: 4,5m min.: 0,5m
Farbdaten	Format: 4:3 Auflösung: 640 x 480 @ 30fps 1280 × 1024 @ 15fps	Format: 16:9 Auflösung: 1920 x 1080 @ 30 fps
Tiefendaten	Auflösung: 320 x 240	Auflösung: 512 x 424
Skelettverfolgung	erfasste Personen: 2 Gelenkpunkte: 20	erfasste Personen: 6 Gelenkpunkte: 25
Mikrofone	4 Mikrofone mit 16 kHz	4 Mikrofone mit 48 kHz
Latenz	Verarbeitung: ~ 90 ms	Verarbeitung: ~ 60 ms
Anschluss	USB 2.0	USB 3.0
Neigungsmotor	nur Vertikal	kein Neigungsmotor

Tabelle 2-1: KINECT und KINECT v2 im Vergleich

3 Grundlagen einer KINECT - Anwendung

Als Grundlage für die Realisierung dieser KINECT Anwendung dient das von Microsoft zur Verfügung gestellte SDK 2.0 für KINECT. Dieses Kapitel gibt einen Überblick über die bisher erschienen SDK - Versionen sowie die neuen Features des KINECT SDK 2.0. Abschließend wird noch kurz auf die Entscheidung über die verwendete Programmiersprache eingegangen.

3.1 Windows Developer Toolkit für KINECT

Das KINECT SDK dient zur Entwicklung eigener KINECT-Anwendungen. Kontinuierlich weiterentwickelt erschien im Oktober 2014 explizit für die KINECT v2 das „KINECT for Windows SDK 2.0“. Das KINECT for Windows SDK 2.0 ist das offizielle Entwickler-Kit, um Anwendungen für KINECT v2 für Windows zu entwickeln.

Bevor gezielt auf die Besonderheiten des KINECT SDK eingegangen wird, noch kurz die Klärung: „Was ist ein SDK?"

„Ein Software Development Kit (SDK) ist ein Werkzeugsatz für Softwareentwickler, das einen einfachen Einstieg in ein spezielles Betriebssystem oder eine Programmiersprache bietet. Ein SDK enthält alle Informationen und Dokumentationen sowie Werkzeuge, die die Software-Entwicklung erleichtern. Dazu gehören Bibliotheken mit Routinen für einen speziellen Computer, ein Betriebssystem oder eine Programmiersprache. Softwareentwickler können mit diesen Bausteinen Anwendungsprogramme modulartig aufbauen.“ [SDK_3-1]

Als die KINECT 2007 für die Xbox 360 erschien, hatten viele Programmierer Ideen für eigene Anwendung. Leider war es anfänglich nur über aufwendige Hacks möglich, den KINECT Sensor am PC zum Laufen zu bekommen. Microsoft erkannte darin einen neuen Markt für die KINECT und entwickelte daraufhin ein KINECT-SDK.

Mit der Version 1.0 brachte man im Vergleich zur aktuellsten Version nur die Grundvoraussetzungen mit. So enthielt das SDK die Treiber für die Verwendung am Computer unter Windows 7, den Zugriff auf die Programmierschnittstellen (API) und einige Quellcode Beispiele. [SDK_3-2]

Im Mai 2012 erschien die Version 1.5, mit signifikanten Verbesserungen, auf dem Markt. Das SDK 1.5 lieferte den Near- und Seated Mode der ersten KINECT Generation. Damit war nun eine Skelettverfolgung im Nahbereich sowie sitzend möglich. Neben noch einigen kleinen Verbesserungen im Bereich Bildqualität, Laufzeit und neuer Sprachen für die Spracherkennung, wurde auch das KINECT Studio eingeführt.

Das KINECT Studio (Abb. 3-1) ist ein Tool, das das Aufnehmen und Wiedergeben von KINECT Daten für die Entwicklungshilfe ermöglicht. Zum Beispiel kann ein Entwickler einen Clip von den Benutzern in der Zielumgebung aufnehmen und dann den Clip zu einem späteren Zeitpunkt für Entwicklungs- und Testzwecke wiedergeben [SDK_3-3]. Es folgten die SDK Versionen 1.5.1 und 1.5.2 mit lediglich geringfügigen Verbesserungen.

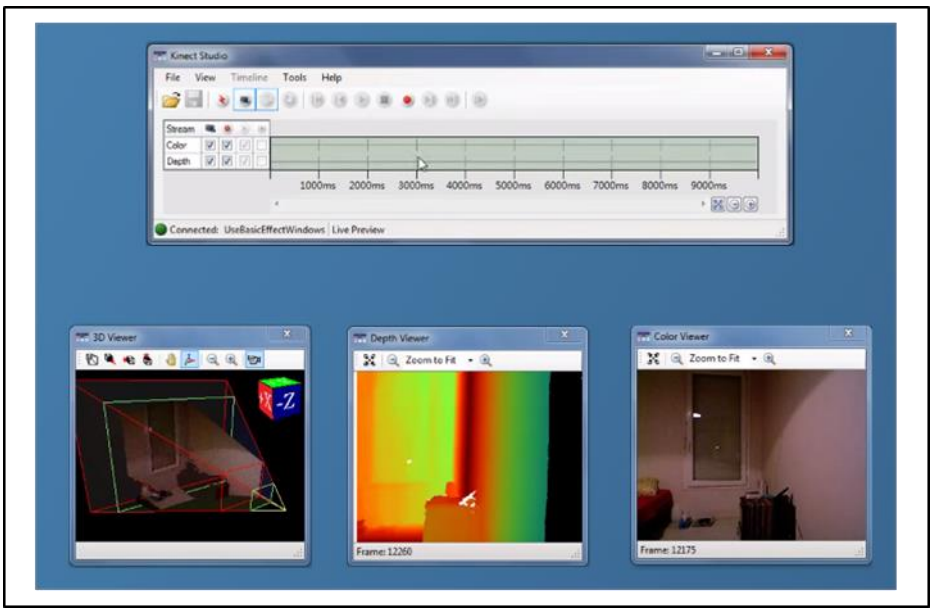


Abbildung 3-1: KINECT Studio aus KINECT SDK 1.5

Mit der Version 1.6 erhielt das SDK neben Erweiterung im Bereich Farbkamera, Tiefen- und Accelerometerdaten die Unterstützung für Windows 8 und Visual Studio 2012. Hinzu kam die Aktualisierung des KINECT Studio. Eine weitere große Erneuerung brachte die Version 1.7, mit der Einführung des KINECT Fusion.

KINECT Fusion (Abb. 3-2) bietet die Möglichkeit des 3D-Objekt-Scans und der Modellierung. Aus dem Tiefenbild der KINECT Kamera produziert das KINECT Fusion in wenigen Sekunden ein realistisches 3D-Abbild. Der Benutzer ist damit in der Lage, aus einer statischen Szene ein detailliertes 3D-Modell dieser Szene zu erzeugen [SDK_3-4].

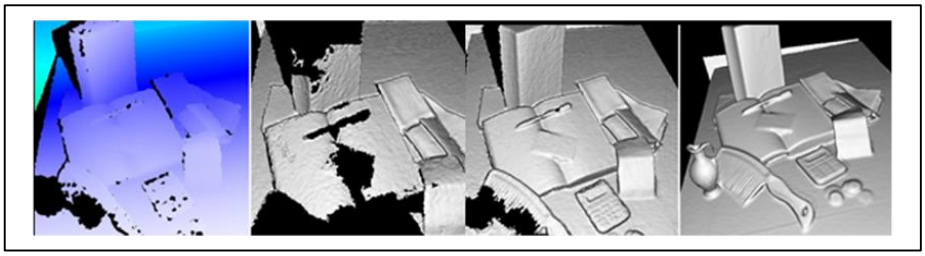


Abbildung 3-2: KINECT Fusion

Mit der finalen Version 1.8 des SDK's, für die erste Generation der KINECT brachte Microsoft im Oktober 2013 noch die Funktion der Hintergrundentfernung auf den Markt. Wie

aus Film und Fernsehen bekannt, entfernt die „Greenscreen“-Funktion den Hintergrund aus der Szene. [SDK_3-5]

Um auf alle Neuerungen der neuen KINECT Generation einzugehen, war es notwendig das SDK komplett zu überarbeiten. So erschien das KINECT for Windows SDK 2.0 nicht nur im neuen Design, sondern auch mit wichtigen Verbesserungen und neuen Funktionen.

Im Vergleich zur Version 1.8 wurden aufgrund der neuen hardwarespezifischen Möglichkeiten der KINECT v2 einige Grundeigenschaften verbessert. Das SDK 2.0 bringt die Möglichkeiten mit, auf das größere Sichtfeld der neuen KINECT einzugehen. Hinzu kommen die Unterstützungen der Full-HD Auflösung der Farbkamera, die dreifach verbesserte Tiefenwahrnehmung und die Erkennung der 25 Skelettpunkte sowie der Handerkennung. Neben weiteren Verbesserungen im Bereich Sprach- und Gesichtserkennung sind auch die Tools KINECT Fusion und KINECT Studio komplett überarbeitet worden.

Als neues Feature hinzugekommen ist der „Visual Gesture Builder“. Er ermöglicht das Erstellen eigener Gesten. Da dieses Tool für die spätere Realisierung noch relevant wird, erfolgt ein ausführlicher Einblick im Punkt 3.2.

Das KINECT for Windows SDK 2.0 unterstützt alle Computer mit Windows 8, 8.1 und neuerdings auch Windows 10. Darüber hinaus besteht eine Verbindung zum Windows - Store. Damit besteht die Möglichkeit die entwickelten Anwendungen direkt im Windows - Store zu veröffentlichen [SDK_3-6].

3.2 Visual Gesture Builder

Der „Visual Gesture Builder“ (VGB) ist ein zusätzliches Tool aus dem KINECT for Windows SDK 2.0 und dient dazu, eigene Gestenbewegungen zu erstellen. Das Tool arbeitet dabei datengesteuert und nutzt maschinelle Lern-Algorithmen, um Gesten zu identifizieren.

Der größte Vorteil des VGB besteht darin, eigene Gesten einfach und grafisch zu erzeugen, ohne jeglichen Quellcode geschrieben zu haben. Der „Visual Gesture Builder“ unterscheidet dabei zwischen den zwei Gestenarten: kontinuierlichen und diskreten Gesten.

Unter „kontinuierlichen Gesten“ versteht man beständig zu ändernde Gesten, z. B. Winken. Diskrete Gesten sind im Gegensatz dazu permanent gleichbleibend, z. B. Gebärdensprache. Je nach Gestentyp arbeitet der VGB nach unterschiedlichen Prinzipien. Kontinuierliche Gesten werden mittels „RFRProgress“ erfasst, die diskreten hingegen nach dem „AdaBoostTrigger“. [VGB_3-7]

Der RFRProgress (Random-Forest-Regression-Progress) nutzt den Random-Forest-Algorithmus, um den Fortschritt der kontinuierlichen Geste zu bestimmen. Random-Forest ist

eigentlich ein Klassifikationsverfahren, welches aber auch zur Regression eingesetzt wird. Es verwendet mehrere verschiedene, nicht zusammenhängende „Entscheidungsbäume“. Für die Bestimmung darf jeder Baum eine Entscheidung treffen und die Klasse mit den meisten Stimmen entscheidet die endgültige Klassifikation [RFR_3-8].

Das AdaBoostTrigger Prinzip dient zur Erkennung eines diskreten Ergebnisses. Es verwendet einen Adaptive Boosting (AdaBoost) Algorithmus für die Bestimmung, ob ein Benutzer eine Geste ausführt. Dabei kann der Wert, anders als bei RFRProgress, nur zwei Zustände haben: ausgeführt oder nicht ausgeführt [ADA_3-9].

Die Benutzeroberfläche des Visual Gesture Builder (Abb. 3-3) besteht aus unterschiedlichen Fenstern und Menüoptionen. Auf der linken Seite wird die Projektstruktur angezeigt. Eine Gestendatenbank, eine sogenannte „Solution“, besteht aus unterschiedlichen Projekten. Jedes Projekt einer Solution entspricht einer einzelnen Geste. Es wird zusätzlich in einen Trainings- und Analyse - Bereich unterteilt. Das Ansichtsfenster in der Mitte zeigt eine 2D und 3D Sicht des Videos, das gerade bearbeitet wird. Unterhalb befindet sich das Steuerungsfenster. Auf der rechten Seite findet man noch die Eigenschaften des Projekts oder der Aufnahmen.

Wie man genau vorgeht, bei der Erstellung einer Geste, wird im Kapitel 5 „Realisierung der prototypischen Lösung“ erklärt.

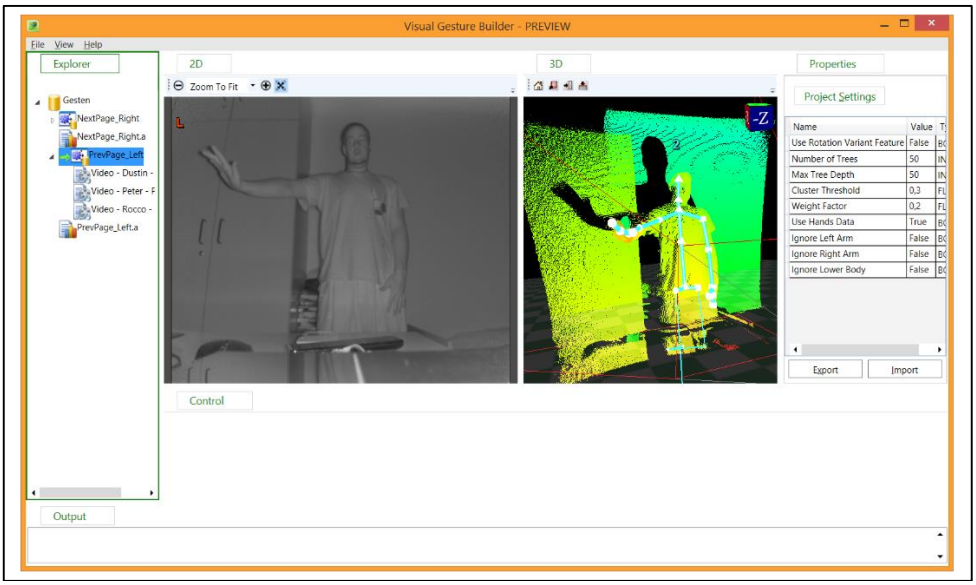


Abbildung 3-3: Visual Gesture Builder

3.3 C#

Das KINECT for Windows SDK 2.0 unterstützt für die Realisierung eigener KINECT Anwendungen jede .NET-Programmiersprache [SDK_3-10]. Zu diesen Programmiersprachen zählen auch die Sprachen C ++, C# oder Visual Basic. Eine Auflistung, zu den restlichen Programmiersprachen die .NET unterstützt, findet man unter [WIKI_3-11].

Es macht aber den Eindruck, dass die drei genannten Programmiersprachen aktuell die gängigsten Sprachen sind, wenn es um die Umsetzung von KINECT-Anwendungen geht.

Aus diesem Grund empfahl es sich die Wahl zwischen C++, C# und Visual Basic zu treffen. Aufgrund der gezielten Unterstützung des .NET Frameworks und der Ähnlichkeit zur Programmiersprache Java, fiel schlussendlich die Wahl auf C#.

C# (C-Sharp) ist eine von Microsoft entwickelte Programmiersprache. Sie ist einfach strukturiert aber sehr leistungsfähig. Da sich schon unzählige Autoren mit diesem Thema beschäftigt haben und es den Rahmen dieser Arbeit sprengen würde, auf alles Wissenswerte zu C# einzugehen, kann sich wer mehr zum Thema C# wissen möchte, beispielsweise im Eric Gunnerson Buch „C# - Die neue Sprache für Microsoft .NET Plattform“ weiter informieren. [GUN00]

4 Entwurf

In diesem Kapitel wird das Konzept und die Anforderungen an die prototypische Lösung erarbeitet. Zu Beginn wird auf die Programmierung der ersten KINECT Version und die Probleme bei der Umstellung auf die KINECT v2 eingegangen. Anschließend werden die Möglichkeiten, die für die Realisierung der Steuerung denkbar waren, dargestellt. Nach Festlegung der Vorgehensweise wird genauer auf die prototypische Lösung eingegangen, dabei werden die Anforderungen an das Tool spezifiziert, welche für die Verwendung einer Präsentation nötig sind, und wie folglich die Umsetzung geplant ist.

4.1 KINECT Programmierung

Für den Einstieg in die komplette Thematik der Arbeit wurde sich zunächst mit der KINECT Programmierung auseinandergesetzt. Dabei wurde als erstes die Programmierung der ersten KINECT Version betrachtet.

Dies hatte folgende Gründe:

1. Etabliertes System auf dem Markt
2. Fachbücher zum Thema „KINECT Programmierung“ auf dem Markt
   z.B. „Microsoft KINECT – Programmierung des Sensorsystems“ [HAN13] oder „KINECT for Windows SDK Programming Guide“ [JAN12].
3. Verfügbare Online - Plattformen
   z.B. „OpenKinect“ [OPEN_4-1]
4. Verfügbare Quellcode Beispiele, Projekte und weitere Hilfen
5. Unwissenheit, dass ein programmiertechnischer Unterschied zwischen KINECT und KINECT v2 besteht. (Auf diese Thematik wird im Punkt 4.2 noch detailliert eingegangen)

4.1.1 1. KINECT Anwendung

Nach kurzer Einarbeitungszeit in die Thematik sollte das Ziel der ersten KINECT Anwendung sein, den Farbdatenstrom der KINECT auszulesen. Also eine Art „Webcam“ - Bild zu erstellen. Für die Entwicklung dieser Anwendung wurde das KINECT SDK 1.6 installiert. Dies kann unter *http://www.microsoft.com/en-us/download/details.aspx?id=34808* heruntergeladen werden. Informationen über die Installation sind ebenfalls auf dieser Seite zu finden. Als Entwicklungsumgebung diente Microsoft Visual C# 2010 Express.

Die Entwicklung der KINECT Anwendung erfolgt als WPF-Anwendung. In der Regel entstehen KINECT Anwendungen in Form von WPF-Anwendungen. Das liegt unter anderem

daran, dass die „PictureBox“ von WPF-Anwendungen sehr flexibel ist und mit fast alle gesendeten Daten des KINECT Sensor ohne Konvertierung zurechtkommt.

Zunächst wurde die DLL „Microsoft.Kinect“ dem Projekt hinzugefügt. Diese steht nach Installation der KINECT SDK zur Verfügung. Danach wurde in der „MainWindows.xaml“ ein 640x480 großes Image-Steuerelement erstellt, welches später für Anzeige des Bildes dient.

```
<Window x:Class="KinectWPF1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="536" Width="770"
       Closing="Window_Closing">
    <Grid>
        <Image Name="image1"
               Height="480" Width="640"
               HorizontalAlignment="Left" VerticalAlignment="Top"
               Stretch="Fill"/>
    </Grid>
</Window>
```

Quellcode 4-1: Image – Steuerelement

Anschließend wurden in der „MainWindows.xmal.cs“ drei globale Variablen deklariert. Neben der Sensorreferenz „KinectSensor“ und einem Ablage-Array „array“, für den Datenspeicher, wurde noch eine Variable für eine WriteableBitmap-Instanz erzeugt. Dabei handelt es sich um eine spezielle Bitmap Klasse von WPF - Anwendungen, die den direkten Zugriff auf die enthaltenen Farbdaten erlaubt.

```
public partial class MainWindow : Window
{
  KinectSensor sensor;
  byte[] array;
  WriteableBitmap bitmap;
```

Quellcode 4-2: Deklaration der benötigten Variablen

Der nächste Punkt war die Initialisierung im Konstruktor. Da es möglich ist, mehrere KINECT Sensoren an einen PC zu verwenden, muss dies in jeder KINECT-Anwendung berücksichtig werden. Dafür gibt es ein „KinectSensor“ - Array, das alle erkannten KINECT Sensoren enthält. Auf MSDN [FOR_4-2] findet man diesbezüglich eine einfache Lösung, welche mittels einer „ForEach“ - Schleife das Array durchläuft und den ersten betriebsbereiten Sensor verwendet.

Danach konnte der Farbdatenstrom der KINECT erfasst werden. Die KINECT liefert natürlich mehrere Ströme, beispielsweise noch den Tiefendatenstrom. Da für diese Anwendung der Farbdatenstrom reicht, wird nur der „ColorStream“ verwendet und durch den Aufruf, der „Enabled“ - Methode aktiviert. Nun konnte der „ColorImageFormat“ - Parameter übergeben werden. Dieser legt das Datenformat, die die Bildinformationen haben sollen, fest. In diesem Beispiel 640 x 480 Pixel bei 30 Frames pro Sekunde. Dies ist das

Standardformat der ersten KINECT Generation. Seit dem SDK 1.6 wäre aber theoretisch auch eine höhere Auflösung möglich. Mit Veröffentlichung des SDK 1.6 ist eine Auflösung von 1280 x 960 Pixeln möglich, jedoch auf Kosten der Framerate. Diese wäre bei Verwendung der höheren Auflösung nur noch bei 10 bis 12 Frames pro Sekunde. [SDK_4-3]

Neben der Initialisierung des Ablage-Arrays wurde auch die Methode „SensorColorFrameReady direkt aufgerufen, welche im nächsten Schritt erzeugt wurde. Zu guter Letzt wurde im Konstruktor mit „sensor.Start()“ der KINECT Sensor gestartet.

```
public MainWindow()
{
   InitializeComponent();
   foreach (var potentialSensor in KinectSensor.KinectSensors)
   {
     if (potentialSensor.Status == KinectStatus.Connected)
     {
        sensor = potentialSensor;
        break;
     }
   }

   if (sensor != null)
   {
     sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
     array = new byte[this.sensor.ColorStream.FramePixelDataLength];
     bitmap = new WriteableBitmap(this.sensor.ColorStream.FrameWidth,
                    this.sensor.ColorStream.FrameHeight, 96.0, 96.0, Pixel-
                    Formats.Bgr32, null);
     image1.Source = bitmap;
     sensor.ColorFrameReady += this.SensorColorFrameReady;
     this.sensor.Start();
   }
}
```

Quellcode 4-3: Initialisierung im Konstruktor

Der nächste Schritt war die Erzeugung der „SensorColorFrameReady“ - Methode. Hier musste zunächst eine „ColorImageFrame“-Instanz erschaffen werden. Das „ColorImageFrame“ ist ein „Buffer“ für den Farbdatenstrom des Sensors.

Ein „Buffer“ stellt die Methoden zum Kopieren von Bytes aus einem Array in ein anderes Array zur Verfügung. Diese Funktion wird mittels „CopyPixelDataTo“ verwendet. Anschließend wurde die „WritePixels“-Methode des „WriteableBitmap“ verwendet, um die Daten direkt in die Bitmap zu übernehmen.

Die Übergabe der Farbdaten an das Image - Steuerelement erfolgt automatisch, da im Konstruktor die Bitmap zugewiesen wurde. Wenn die Bitmap nun verändert wird, greift sich das Image-Steuerelement automatisch den neuen Inhalt. Eine manuelle Freigabe des Frameobjektes entfällt, da es durch die Using-Struktur von selbst erfolgt.

```
private void SensorColorFrameReady(object sender,
       ColorImageFrameReadyEventArgs e)
{
  using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
  {
    if (colorFrame != null)
    {
      colorFrame.CopyPixelDataTo(array);
      bitmap.WritePixels(new Int32Rect(0, 0, bitmap.PixelWidth,
            bitmap.PixelHeight),
            array,
            bitmap.PixelWidth * sizeof(int),
            0);
     }
   }
 }
```

Quellcode 4-4: SensorColorFrameReady - Methode

Zum Abschluss musste noch eine „Window_Closing“ - Methode erzeugt werden. Diese dient dazu, dass beim Schließen der Anwendung der KINECT Sensor ebenfalls geschlossen wird. Dies erfolgt durch die Funktion „sensor.Stop()“. Das Beenden des KINECT Sensors ist wichtig, um diesen für andere Programme oder eine spätere neue Instanz der eigenen Anwendung wieder freizugeben.

```
private void Window_Closing(object sender,
       System.ComponentModel.CancelEventArgs e)
{
   sensor.Stop();
}
```

Quellcode 4-5: Window_Closing - Methode

Damit ist die Programmierung dieser KINECT-Anwendung abgeschlossen. Das Ergebnis (Abb. 4-1) dieser Anwendung ist wenig spektakulär und auch nicht sehr überraschend. Die Anwendung zeigt uns das Farbbild vor der Kamera.

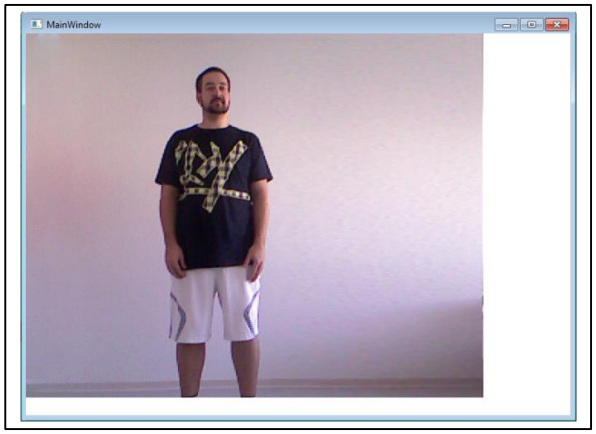


Abbildung 4-1: Ergebnis der 1. KINECT Anwendung

4.1.2 Programmierung der Skelettverfolgung

Für die Realisierung der Gestensteuerung wird die Skelettverfolgung der KINECT benötigt. Daher war der nächste Schritt, diese umzusetzen. Als Basis diente die vorherige KINECT Anwendung. Der Grundaufbau dieser KINECT Anwendung blieb weitestgehend unverändert und wurde nur um die Funktionen der Skelettverfolgung erweitert. So wurde beispielsweise keine Änderung an der „MainWindow.xaml“ vorgenommen.

Als Erstes musste eine zusätzliche Variable von Typ „Skeleton[]“ deklariert werden, welche später die Skelettdaten enthält. Im Konstruktor wurde der hinzugefügte Farbdatenstrom durch den Skelettdatenstrom „SkeletonStream“ ersetzt und wie beim Farbdatenstrom mit der Methode „Enable()“ aktiviert. Die Methode „SensorColorFrameReady“, welche für das Erstellen des Bildes zuständig war, wird nicht mehr benötigt. Diese wurde durch die Methode „SensorSkeletonFrameReady“ ersetzt. Der restliche Konstruktor blieb unverändert.

```
if (sensor != null)
{
  //sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
  sensor.SkeletonStream.Enable();
  skeletonArray = new Skeleton
  [this.sensor.SkeletonStream.FrameSkeletonArrayLength];

  array = new byte[this.sensor.ColorStream.FramePixelDataLength];
  bitmap = new WriteableBitmap(this.sensor.ColorStream.FrameWidth,
  this.sensor.ColorStream.FrameHeight, 96.0, 96.0, PixelFormats.Bgr32,null);
  image1.Source = bitmap;

  //sensor.ColorFrameReady += this.SensorColorFrameReady;
  sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;
  this.sensor.Start();
}
```

Quellcode 4-6: Änderungen im Konstruktor für Skelettverfolgung

Bei der Erzeugung der „SensorSkeletonFrameReady“ - Mehtode wurde ebenfalls zunächst wieder ein Buffer erzeugt. In diesen Fall lautet dieser „SkeletonFrame“.

Nun folgte der große Unterschied zur vorherigen Anwendung. Da diesmal nicht nur das Kamerabild ausgegeben werden sollte, sondern die Skelettdaten des Nutzers, mussten diese selbst gezeichnet werden. Dazu wurde zunächst ein „DrawingVisual“ erstellt, die als Zeichenfläche diente. Um in der „DrawingVisual“ zu zeichnen, musste von der „DrawingVisual“ ein „DrawingContext“ abgeleitet werden.

```
private void SensorSkeletonFrameReady(object sender,
      SkeletonFrameReadyEventArgs e)
{
  using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
  {
      ...
  }
  BitmapSource bs = BitmapSource.Create(640, 480, 96, 96,
      PixelFormats.Bgr32, null, array, 640 * 4);
  DrawingVisual drawingVisual = new DrawingVisual();
  DrawingContext drawingContext = drawingVisual.RenderOpen();
  drawingContext.DrawImage(bs, new Rect(0, 0, 640, 480));
```

Quellcode 4-7: Erstellung der Zeichenfläche in der SensorSkeletonFrameReady - Methode

Nun erfolgte der Aufbau der Skelettdaten. Dies entstand ebenfalls in der „SensorSkeleto-FrameReady“ - Methode. Die KINECT der ersten Generation besitzt 20 Gelenkpunkte. Diese mussten nun alle miteinander verbunden werden. Das ist zwar etwas umständlich, aber nicht besonders kompliziert. Man nimmt sich einen Gelenkpunkt aus dem Skelett - Array und zeichnet es Knochen für Knochen auf dem Bildschirm. Die Gelenkpunkte (Joints engl. für Gelenke) und deren Koordinaten erhält man aus dem Joints-Array. Eine Auflistung aller Gelenkpunkte findet man auf MSDN [JOI_4-4].

```
foreach (Skeleton aSkeleton in skeletonArray)
{
// Linker Arm
 DrawBone(aSkeleton.Joints[JointType.HandLeft],
       aSkeleton.Joints[JointType.WristLeft], pen, drawingContext);
 DrawBone(aSkeleton.Joints[JointType.WristLeft],
       aSkeleton.Joints[JointType.ElbowLeft], pen, drawingContext);
 DrawBone(aSkeleton.Joints[JointType.ElbowLeft],
       aSkeleton.Joints[JointType.ShoulderLeft], pen, drawingContext);
 DrawBone(aSkeleton.Joints[JointType.ShoulderLeft],
       aSkeleton.Joints[JointType.ShoulderCenter], pen, drawingContext);

// Rechter Arm
 ...
 }
```

Quellcode 4-8: Aufbau der Skelettdaten am Bsp. Linker Arm

Das eigentliche Zeichnen erfolgt allerdings in einer weiteren Funktion, der „drawBone“ - Methode. Mittels der „MapSkeletonPointToColorPoint“ - Funktion werden die dreidimensionalen Skelettkoordinaten zum Zeichnen in zweidimensionale Koordinaten umgewandelt. Dabei wird zwischen den drei Verfolgungszuständen der Skelettverfolgung unterschieden. In Abhängigkeit der Tiefendaten und des Skelettdatenstroms können die Gelenkpunkte vollständig verfolgt sein (Tracked), anhand von anderen Gelenkpunkten abgeleitet bzw. vermutet werden (Inferred) oder nicht verfolgt sein (NotTracked). Je nach Verfolgungszustand wird eine durchgehende Linie (Tracked), eine gestrichelte Linie (Inferred) oder keine Linie gezeichnet (NotTracked). Anschließend wurde der „DrawingContext“ verwendet, um die entsprechende Linie zwischen den beiden Gelenkpunkten zu zeichnen.

```
private void DrawBone(Joint jointFrom, Joint jointTo, Pen aPen,
       DrawingContext aContext)
{
  if (jointFrom.TrackingState == JointTrackingState.Tracked ||
       jointTo.TrackingState == JointTrackingState.Tracked)
  {
     ColorImagePoint p1 = sensor.CoordinateMapper.
              MapSkeletonPointToColorPoint(jointFrom.Position,
              ColorImageFormat.RgbResolution640x480Fps30);
     ColorImagePoint p2 = sensor.CoordinateMapper.
              MapSkeletonPointToColorPoint(jointTo.Position,
              ColorImageFormat.RgbResolution640x480Fps30);
      aPen.DashStyle = DashStyles.Solid;
      aContext.DrawLine(aPen, new Point(p1.X, p1.Y), new Point(p2.X, p2.Y));
   }
   ...
}
```

Quellcode 4-9: DrawBone – Methode

Zum Schluss mussten die Daten noch ins Image-Steuerelement geschrieben werden. Nach dem der „DrawingContext“ mittels „Close()“ beendet wurde, kommt die Klasse „RenderTargetBitmap“ zum Einsatz. Diese dient dazu, das erzeugte Visual - Objekt in eine Bitmap umzuwandeln, welches abschließend an das Image-Steuerelement aus der „MainWindow.xaml“ übergeben wird.

Als Ergebnis (Abb. 4-2) wird das Skelett des Nutzers angezeigt und verfolgt dessen Bewegungen.

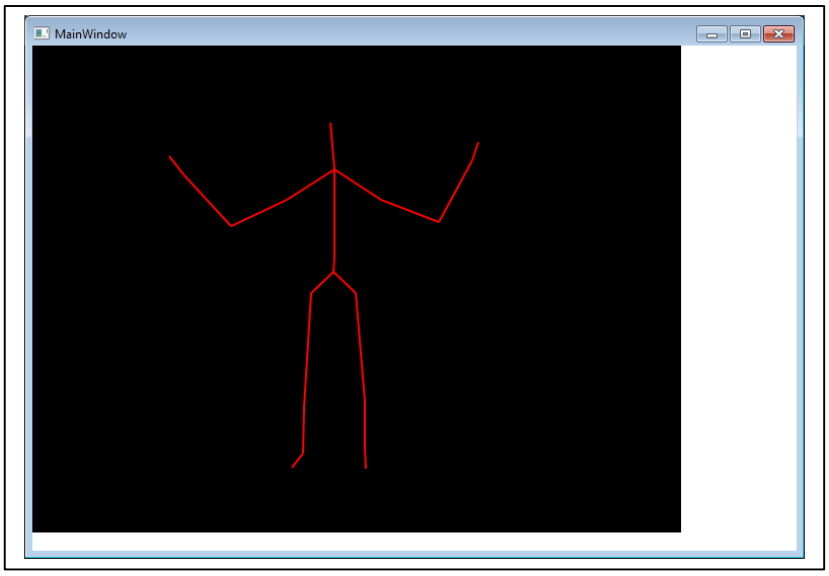


Abbildung 4-2: Ergebnis Skelettverfolgung KINECT v1

4.2 Probleme bei der Verwendung von KINECT v2

Die Umsetzung der prototypischen Lösung dieser Arbeit soll mittels KINECT v2 erfolgen, weil diese mit ihren technischen Möglichkeiten der ersten KINECT weit überlegen ist. Da die Skelettverfolgung funktionierte, sollte nun die Realisierung mit der KINECT v2 beginnen.

Nach Installation des KINECT for Windows SDK 2.0, Download und Installationsanweisungen unter *http://www.microsoft.com/en-us/download/details.aspx?id=44561* und der Verwendung der neuen Entwicklungsumgebung Microsoft Visual Studio 2013, wurde leider festgestellt, dass die programmierte KINECT-Anwendung nicht mehr funktionierte.

Das SDK 2.0 erhielt nicht nur neue Features, sondern auch eine komplett neue Architektur. KINECT-Anwendungen der ersten Generation sind daher nicht mit der neuen KINECT kompatibel. KINECT-Anwendungen müssen erst an die neue Architektur angepasst werden.

4.2.1 Neue KINECT Architektur

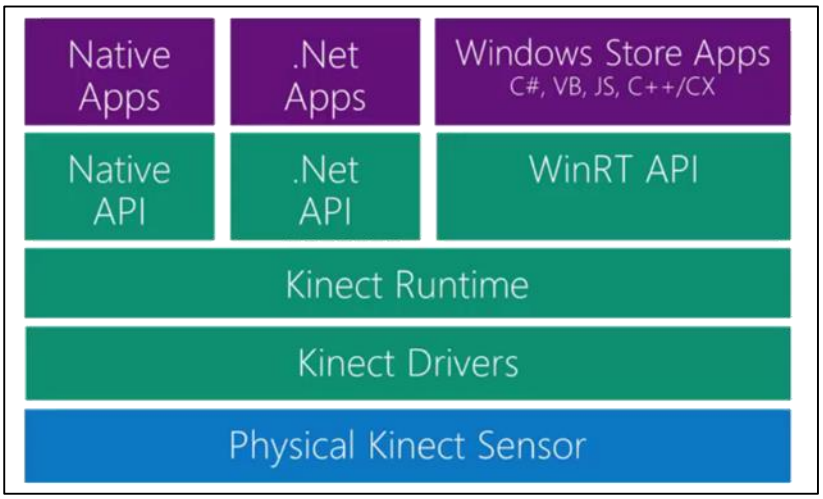


Abbildung 4-3: KINECT for Windows SDK 2.0 Architektur

Wenn man sich die Softwarearchitektur (Abb. 4-3) der neuen KINECT als Schichtenmodell betrachtet, befindet sich auf der untersten Schicht der physikalische KINECT Sensor, mit allen seinen Hardwarekomponenten. Der Farb- und Tiefenkamera, den Mikrofonen sowie der USB-Adapter, durch den der KINECT Sensor mit dem Computer verbunden ist.

Auf der nächsten Schicht befinden sich die KINECT Treiber, die ebenfalls vom SDK mitgeliefert werden. Darüber ist die KINECT Runtime. Hier geschieht die Datenverarbeitung der verschiedenen Komponenten, beispielsweise die Verarbeitung der Skelettverfolgung. Bis zu diesem Punkt unterscheidet sich die Architektur, zur vorherigen Version, prinzipiell nicht. Erst mit der Unterstützung der APIs unterscheiden sich die Architekturen.

Wo bei der ersten KINECT nur Zugriff auf die „NativeAPI“ bestand, bietet das SDK 2.0 Zugriff auf drei verschiedene APIs. Neben der ebenfalls vorhanden Native API, steht noch die „.NET API“ und die „Windows - Runtime API“ bereit, die die Entwicklung von Windows Store Anwendungen ermöglicht [SDK_4-5].

4.2.2 Anpassung der KINECT Anwendung

Durch die Treiberarchitektur der ersten KINECT war es notwendig, bei der Programmierung die Verwendung von mehrere KINECT Sensoren an einem PC zu berücksichtigen. Im neuem SDK gibt es diese Unterstützung nicht mehr, weshalb die „ForEach“-Methode nicht mehr benötigt wird. Nun reicht es im Konstruktor, einen Zeiger auf den verwendeten Sensor zu erstellen.

```
public partial class MainWindow : Window
    {
        private KinectSensor kinectSensor = null;

        public MainWindow()
        {
            this.kinectSensor = KinectSensor.GetDefault();
            this.kinectSensor.Open();
            this.DataContext = this;
            InitializeComponent();
        }
```

Quellcode 4-10: Erstellung des Zeigers auf KINECT Sensor

Wenn man einen Datenstrom der Kinect z.B. den Farbdatenstrom anzapfen wollte, registrierte man diesen an einen „FrameReady“-Event. Bei der Kinect v2 ist die Lage aufgrund der neuen Treiberarchitektur etwas anders. Frames werden nun über agierende „FrameReader“-Instanzen an ihre Verbraucher weitergeleitet.

```
//KINECT v1
sensor.ColorFrameReady += this.SensorColorFrameReady;

//KINECT v2
this.colorFrameReader.FrameArrived += this.Reader_ColorFrameArrived;
```

Quellcode 4-11: Unterschied des Event - Handling

Hinzu kommt, dass die erste KINECT drei Datenströme lieferte: den Farbdatenstrom (ColorStream), den Tiefdatenstrom (DepthStream) und den für die Skelettdaten (SkeletonStream). Zum Verwenden einer dieser Ströme wurde das zugehörige „FrameReady“ - Event hinzugefügt. Wollte man jedoch zwei oder alle drei Ströme nutzen, musste man nicht jedes einzige „FrameReady“-Event hinzufügen, sondern es gab das “AllFrames-Ready“-Event, welches alle Ströme enthielt. Bei der KINECT v2 steht „AllFramesReady“ nicht mehr zur Verfügung. Daher muss man, bei Verwendung mehrerer Datenströme,

diese sammeln und gemeinsamen in einer Methode gebündelt bearbeiten. Alternative bietet sich die Nutzung des „MultiFrameSourceReader“ an. Für diese KINECT-Anwendung genügt es, wenn man die „FrameArrived“-Methode implementiert.

Speziell bei der Implementierung der Skelettverfolgung sollte man der Verwendung der KINECT v2 beachten, dass diese über eine Erkennung von mehr Gelenkpunkte verfügt. Die Verbindungen zwischen den einzelnen Gelenkpunkten erfolgen auch nicht mehr durch eine „ForEach“-Schleife, welche ein Array mit den Skelettdaten durchläuft, sondern wird über eine „List“ im Konstruktor erfasst. Dabei wird ein Knochen (Bone) als Linie zwischen zwei Gelenkpunkten definiert. Anschließend werden die zugehörigen Gelenkpunkte dem Knochen hinzugefügt.

```
// Knochen definieren
this.bones = new List<Tuple<JointType, JointType>>();

// Linker Arm
this.bones.Add(new Tuple<JointType, JointType>(JointType.ShoulderLeft,
       JointType.ElbowLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.ElbowLeft,
       JointType.WristLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft,
       JointType.HandLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.HandLeft,
       JointType.HandTipLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft,
       JointType.ThumbLeft));

// Rechter Arm
...
```

Quellcode 4-12: Definition der Gelenkpunkte am Bsp. Linker Arm

Wie im Punkt 2.4 dieser Arbeit bereits erklärt, ist die KINECT v2 neben der Erkennung von mehr Gelenkpunkten auch in der Lage, Handzustände zu erkennen. Die KINECT-Anwendung wurde daher um eine „DrawHand“-Methode erweitert. Diese lässt mittels einer unterschiedlich eingefärbten Ellipse, die drei Handzustände visuell anzeigen.

```
// Definition der Farbe der Handzustände
private readonly Brush handClosedBrush = new SolidColorBrush(
       Color.FromArgb(128, 255, 0, 0));
...
// Zeichnen der Handzustände
private void DrawHand(HandState handState, Point handPosition,
       DrawingContext drawingContext)
{
  switch (handState)
  {
    case HandState.Closed:
       drawingContext.DrawEllipse(this.handClosedBrush, null, handPosition,
              HandSize, HandSize);
       break;
    ...
```

Quellcode 4-13: Erfassung des Handzustandes am Bsp. Hand geschlossen

Nach abschließender Implementierung der Methoden „DrawBone“, welche ebenfalls die Unterscheidung zwischen den drei Verfolgungszuständen „Tracked“, „Inferred“ und „Not-Tracked“, die aus der KINECT v1 Programmierung bekannt sind, beinhaltet und der „Main-Window_Closing“ - Methode, erhält man ein ähnliches Ergebnis (Abb. 4-4) der Skelettverfolgung wie bei der vorherigen Version. Diese Version ist neben der Erkennung der zusätzlichen Gelenkpunkte auch in Lage die drei Handzustände zu erkennen.

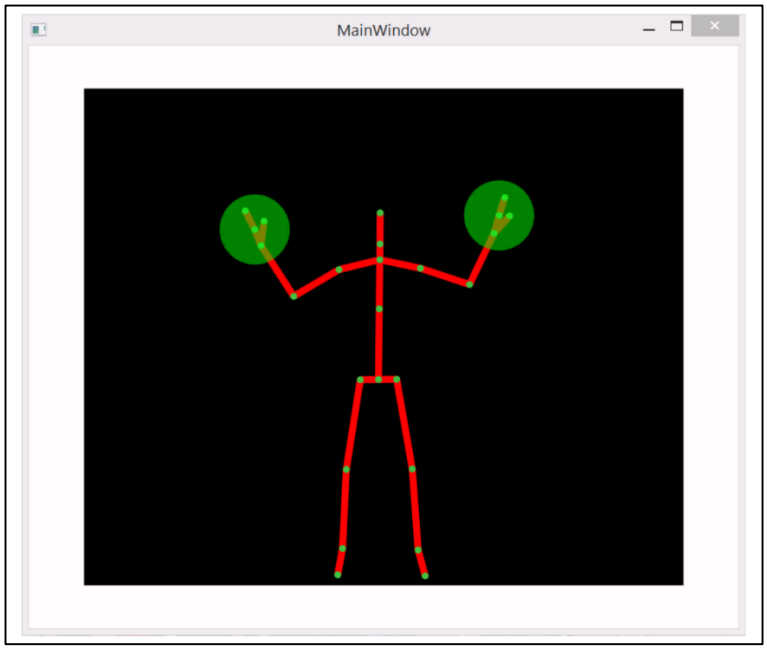


Abbildung 4-4: Ergebnis Skelettverfolgung KINECT v2

4.3 Möglichkeiten zur Realisierung der Gesten

Nachdem nun geklärt ist, wie die Skelettverfolgung unter KINECT v2 funktioniert, war der nächste Punkt, der für die Realisierung geklärt werden musste, die Umsetzung der Gesten. Nach der Einarbeitung in die KINECT Programmierung ergaben sich drei potenzielle Möglichkeiten:

1. mittels Quellcode eigene Gesten erstellen
2. mittels der Basissteuerung der KINECT v2 den „Hand Pointer Gestures“
3. mittels Visual Gesture Builder eigene Gesten erstellen

4.3.1 Eigene Gesten schreiben

Die klassische Methode, eigene Gesten zu erstellen, ist die Möglichkeit diese einfach selbst zu programmieren. Dies war das Standardverfahren bei der ersten KINECT Generation. Dadurch gibt es natürlich mittlerweile viele Code-Beispiele für bereits erstellte Gesten. Diese müssen gegebenenfalls noch an die Bedürfnisse angepasst werden, es steht einem jedoch so schnell eine Geste zur Verfügung.

Meist handelt es sich dabei aber um einfache Standardgesten. Wenn man aber nun wirklich eine individuelle Geste erstellen möchte, ist dies mit viel Aufwand bzw. viel Quellcode verbunden. Für das Erstellen eigener Gesten nutzt man den heuristischen Ansatz. Das bedeutet, die Vorgehensweise zur Erstellung von Gesten ist nur durch ständiges Testen und Optimieren möglich.

Das SDK liefert zwar die Gelenkpunkte, jedoch muss in jeden Fall geprüft werden, wo sich der Gelenkpunkt aktuell befindet. Anschließend muss er mit der vorherigen Position verglichen werden (Abb. 4-5). Dies erfolgt nur mittels eines eigenen geschriebenem Algorithmus, der die relativen Positionen der Gelenkpunkte zueinander berechnet. Daher sollten fortgeschrittene programmier- und mathematische Kenntnisse, bei dieser Art der Gestenerstellung, vorhanden sein.

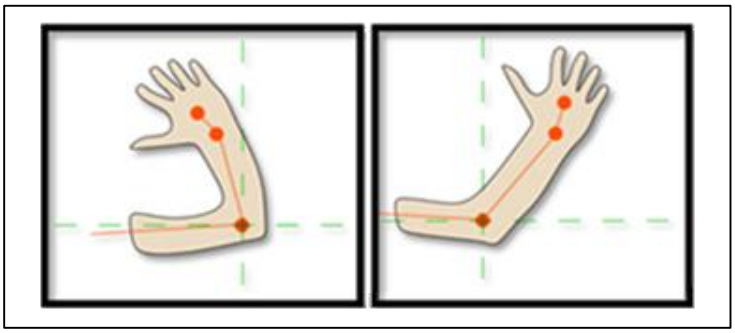


Abbildung 4-5: Verschiebung der Gelenkpunkte bei Ausführung einer Geste

Trotz viel vorhandenen „Beispiel-Code" im Netz ist diese Möglichkeit wohl die aufwendigste zur Erstellung eigener Gesten. Für eine gute funktionierende Geste muss viel Zeit investiert werden, da sehr viel getestet werden muss. Hinzu kommt, dass nach jedem Test die entsprechenden Parameter angepasst werden müssen. Denn selbst die simpelste Geste ist eine sehr komplexe Angelegenheit. Denn jeder Mensch führt Gesten unterschiedlich aus, beispielsweise der Abstand zum Körper. So winken manche Menschen relativ nah am eigenen Körper und bei anderen wiederum ist die Hand beim Winken weiter weg vom Körper. Schon die eigene Körpergröße und die Geschwindigkeit bei der Ausführung der Geste spielt bei der Erstellung von Gesten eine wichtige Rolle.

Wer jedoch die Zeit investiert, kann auch wiederum auch auf ein gutes Resultat zurückschauen. Erstellte Gesten, welche nach diesem Prinzip entwickelt wurden, funktionieren in der Regel gut. Obendrein sind die Gesten auch meist massentauglich, wenn bei der Programmierung auf die unterschiedliche Ausführung von Gesten eingegangen wurde.

4.3.2 Hand Pointer Gestures

Die Funktion des „Hand Pointer Gestures“ bringt die KINECT v2 durch das KINECT for Windows SDK 2.0 mehr oder weniger von Haus aus mit. Dabei handelt es sich nicht konkret um einzelne Gesten, die eine Aktion ausführen, wie bei den anderen beiden Möglich-

keiten. Sondern die „Hand Pointer Gestures“ funktioniert eher wie eine Maus am Computer. Mittels Handbewegung steuert man einen Cursor über den Bildschirm. Durch Heben der rechten oder der linken offenen Hand wird die Funktion aktiviert und der Cursor erscheint auf dem Bildschirm (Abb. 4-6).

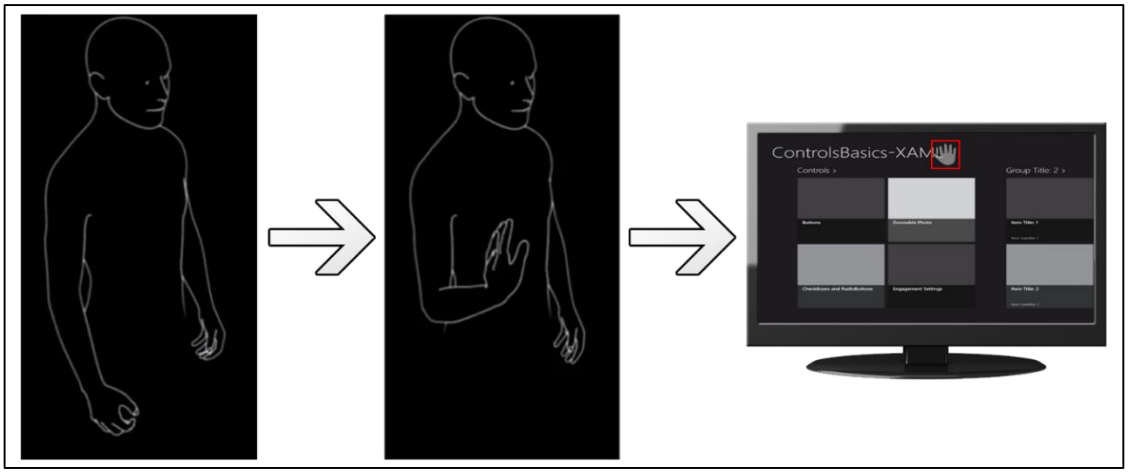


Abbildung 4-6: Aktivierung des „Hand Pointer Gestures“

Mit offener gehobener Hand ist man nun in der Lage, den Cursor an die gewünschte Position zu bewegen, wie mit der Maus des Computers. Dabei kann man den Cursor nicht nur in X- und Y-Richtung manövrieren, sondern aufgrund der Tiefenkamera auch in Z-Richtung (Abb. 4-7). Dies wird beim Betätigen von Buttons, Links etc. interessant, denn durch Bewegung in die Z- Richtung wird ein Klick simuliert.

Abbildung 4-7: Ausrichtung des „Hand Pointer Gestures“

Bereits vorprogrammiert ist sogar ein visuelles Feedback beim Betätigen von Elementen (Abb. 4-8). So wechselt die Farbe des Cursors von Grau auf Weiß und die Kontur des Cursors erscheint.

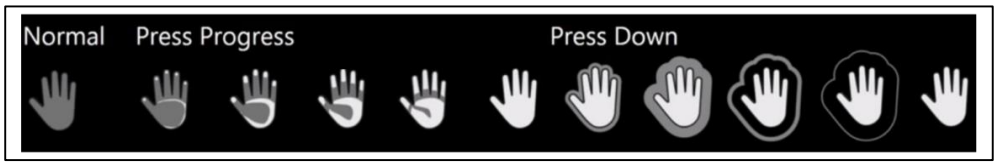


Abbildung 4-8: Visuelles Feedback beim Betätigen von Elementen

Darüber hinaus bieten der „Hand Pointer Gestures“ noch eine „Greif“ - Funktion. Dieser Zustand gestattet es, beispielsweise eine Scroll - Funktion zu simulieren. Dadurch ist es möglich, auf Webseiten zu scrollen. Es muss nur die offene, gehobene Hand geschlossen werden. Im geschlossenen Zustand bewegt sich nun nicht mehr der Cursor, sondern das „gegriffene“ Element. Wie bei dem Betätigen von Elementen ist hier ebenfalls ein visuelles Feedback vorprogrammiert (Abb. 4-9). Die KINECT erkennt, wenn die Hand geschlossen wird und der Cursor wechselt von einer offenen zu einer geschlossenen Hand.

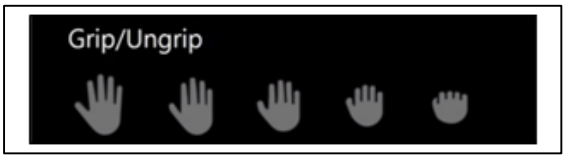


Abbildung 4-9: Visuelles Feedback beim Greifen von Elementen

Auch die Implementierung des „Hand Pointer Gestures“ ist relativ simpel, da diese wie erwähnt mit der SDK mitgeliefert wird.

Grundsätzlich ist die „Hand Pointer Gestures“ - Funktion eine gute Art der KINECT Steuerung. Für die Implementierung werden nicht unbedingt sehr gute Programmierkenntnisse benötigt und auch das Ausführen von Klicks ist dadurch bereits vorhanden. Dennoch ist diese Art der Steuerung für das Ziel dieser Arbeit nicht perfekt geeignet. Der Cursor reagiert auf die Hand sehr gut, manchmal zu gut. Häufig wird während einer Präsentation mit den Händen gestikuliert, um beispielsweise eine Aussage zu verdeutlichen. Dabei kann unabsichtlich der Cursor aktiviert werden. Natürlich ist dies bei den anderen beiden Möglichkeiten auch nicht hundertprozentig ausgeschlossen, jedoch liegt die Wahrscheinlichkeit hier am höchsten. Hinzu kommen die Probleme beim Aktivieren von Button, Links etc.. Grundsätzlich funktioniert das Aktivieren der Elemente sehr gut, aber es hängt auch ein bisschen an der Größe der Schaltfläche. Bei kleineren Elementen kann das Betätigen etwas zur Geduldsprobe werden. Aufgrund der guten Reaktion auf die Hand passiert es schnell, dass man beim Betätigen schon wieder neben die Schaltfläche rutscht und diese dadurch nicht mehr trifft.

4.3.3 Visual Gesture Builder

Die dritte Möglichkeit und nach der kurzen Einarbeitung in die Thematik der KINECT Programmierung, für das Ziel dieser Arbeit, das am besten geeignete Prinzip, ist die Realisierung der Gestensteuerung mittels des „Visual Gesture Builder“. Wie im Punkt 3.2 bereits

vorgestellt, ist der VGB ein Tool, das mit dem KINECT for Windows SDK 2.0 eingeführt wurde.

Der „Visual Gesture Builder“ ermöglicht es anhand von Videos individuelle Gesten zu erstellen. Die Gestaltung von Gesten ist relativ schnell und leicht verständlich. Es sind keine großen Vorkenntnisse nötig. Auch das Einbinden der Gesten in die KINECT Anwendung ist mit wenig Quellcode leicht realisierbar. Ein weiteres großes Plus ist die Realisierung von mehreren Gesten innerhalb eines Projektes.

Die Massentauglichkeit und die Genauigkeit der Erfassung der Gesten während der Ausführung wird der Abschluss dieser Arbeit zeigen. Nach den Tests und der Auswertung im Kapitel 6 wird sich zeigen, ob die Entscheidung, die Realisierung der Gestensteuerung mittels des „Visual Gesture Builder“ vorzunehmen, richtig war.

4.3.4 Vor- und Nachteile der Realisierungsmöglichkeiten

Die nachfolgende Tabelle zeigt noch einmal zusammenfassend die Vor- und Nachteile der drei Möglichkeiten, die für Realisierung der Gesten infrage kamen.

Prinzip	Vorteile	Nachteile
Ausschließliche Programmierung	• mehrere individuelle Gesten möglich • gute funktionierende Gesten realisierbar • Gesten sind massentauglich • Beispiel Quellcode vorhanden	• sehr gute Programmierkenntnisse werden benötigt • viel Quellcode • sehr zeitintensiv • Heuristiken - Ansatz
Hand Pointer Gestures	• leichte Implementierung • visuelles Feedback • vorprogrammierte Buttonsteuerung	• keine Individualität • höchste Wahrscheinlichkeit der unabsichtlichen Aktivierung • bei kleinen Elemente (Button, Links) sehr schwierig diese zutreffen
Visual Gesture Builder	• mehrere individuelle Gesten möglich • einfache und schnelle Realisierung • wenig Quellcode • keine umfangreichen Vorkenntnisse benötigt	• Änderungen bzw. Anpassung der Gesten umständlich • bisher wenig Resonanz bei Verwendung dieses Prinzips vorhanden

Tabelle 4-1: Vor- und Nachteile der Realisierungsmöglichkeiten

4.4 Anforderungen an das Tool

Wie aus dem Anfang dieser Diplomarbeit zu entnehmen, ist das Ziel dieser Arbeit, eine Steuerungsmethode zu entwickeln, die es ermöglicht, eine PowerPoint-Präsentation mittels KINECT zu steuern.

PowerPoint ist ein von Microsoft entwickeltes Computerprogramm, mit dem sich interaktive Präsentationen erstellen und anzeigen lassen. Zurzeit erfolgt die Steuerung der Präsentationen in der Regel durch Maus oder Tastatur. Ob nun Maus oder Tastatur mit beiden lässt sich leicht durch die Präsentation steuern. Ihr großer Nachteil ist jedoch immer: die Nähe zum PC.

Daraufhin wurden sogenannte „Presenter“ eingeführt. Diese ermöglichen es, die Präsentationen auch entfernt vom PC zu steuern. Der Nachteil hierbei ist jedoch, dass eine Präsentation mit Video- oder Musikdatei nur in Kombination mit Maus oder Tastatur möglich ist, da der „Presenter“ nicht die Funktion bietet, eine solche Datei abzuspielen.

Bei der Vielzahl an Funktionen, die PowerPoint bietet, war von Anfang an klar, dass nicht alle Funktionen mittels KINECT realisierbar sind. Es musste klar definiert werden, welche Funktionen werden benötigt und sind auch mittels KINECT lösbar. Als Bezugspunkt für die Entscheidung, welche Funktionen benötigt werden, wurde die Frage gestellt: „**Welche Funktionen werden für eine professionelle Präsentation beansprucht?**“

Nach Recherchen zum Thema „Professionelle Präsentation“ kam die Feststellung, dass sich diese vor allem im Punkt „Schlichtheit“ widerspiegeln. Professionelle Präsentationen enthalten eigentlich keine Effekte und besitzen Folien mit überschaubarem Inhalt mit maximal einem Bild oder Video.

Aus diesen Erkenntnissen definierten sich folgende technische Anforderungen an das Tool:

- Steuerung durch die Präsentation:
  Die wohl logischste als auch wichtigste Funktion ist es, dass das Tool die Möglichkeit bietet, zwischen den Folien zu wechseln. Sowohl der Wechsel zur nächsten Folie, als auch zur vorherigen Folie muss möglich sein.

- Videodatei abspielen:
  PowerPoint bietet die Möglichkeit ein Video in Präsentationen zu integrieren, welches während der Präsentation abgespielt werden kann. Das Tool muss daher die Möglichkeit bieten, dieses Video zu starten.

- Audiodatei abspielen:
  Genauso wie bei Videodateien können auch Audiodateien integriert werden. Das Tool muss auch hier in der Lage sein diese Datei zu starten.

- Hyperlink öffnen:
  Hin und wieder enthalten Präsentation auch Links zu anderen Dokumenten oder Webseiten. Das Tool muss auch diese Möglichkeit bieten, diese zu öffnen.

- Bilder vergrößern/verkleinern:
  Das Tool sollte auch die Möglichkeit bieten, Bilder zu vergrößern bzw. verkleinern.

4.5 Vorstellung an die Realisierung

Nach dem nun die benötigten Funktionen definiert sind und auch die Verwirklichung der Gesten geklärt ist, stellt sich die Frage, wie die Umsetzung der Funktionen durch die Gesten erfolgen kann.

Da sich gegen die „Hand Pointer Gestures“-Steuerung entschieden wurde, wo die Steuerung ähnlich wie bei einer Maus bereits definiert ist, muss eine Möglichkeit gefunden werden, wie eine individuell erstellte Geste eine definierte Funktion, beispielsweise „Nächste Folie“, ausführen kann. Relativ schnell kam die Idee den Tastendruck der Tastatur zu simulieren, wenn die Geste ausgeführt wurde.

Die Tastatur ist eine gängige Methode, um Präsentationen zusteuern. Da es in den meisten Programmiersprachen möglich ist einen Tastendruck zu simulieren, besteht der Plan nun darin, bei Ausführung der Geste die jeweilige Taste bzw. Shortcut der Funktion zu zuordnen. Beispiel „Nächste Folie“, diese Funktion kann durch die Tasten „ENTER“, „LEERTASTE“ oder auch mit der „rechten Pfeiltaste“ ausgeführt werden. Wird nun die Geste für die Funktion „Nächste Folie“ durchgeführt, wird dadurch der Tastendruck der „rechten Pfeiltaste“ simuliert, wodurch die Funktion in der Präsentation ausgeführt wird.

5 Realisierung der prototypischen Lösung

Nach der Einarbeitung in die KINECT Programmierung und der Definition der Anforderungen an das Tool in Kapitel 4, beschäftigt sich dieses Kapitel mit der eigentlichen Umsetzung der prototypischen Lösung. Zunächst wird erklärt, wie Gesten mit dem „Visual Gesture Builder“ erstellt und in eine KINECT-Anwendung eingebunden werden. Danach erfolgt die Umsetzung der Funktionen mittels einem simulierten Tastendruck.

Im abschließenden Punkt dieses Kapitels wird die Realisierung der kompletten KINECT - Anwendung aufgezeigt. Zur Übersichtlichkeit werden wie bereits im vorherigen Kapitel nur Quellcode Auszüge, welche für den aktuellen Punkt wichtig sind, dargestellt. Der vollständige Quellcode zur prototypischen Lösung mit allen benötigten Komponenten (z.B. Gestendatenbank), sowie alle aus dem vorherigen Kapitel realisierten Beispiele befinden sich auf der beiliegenden CD.

5.1 Anpassung der Grundidee für die Realisierung

Die Grundidee der Realisierung war es, die definierten Funktionen:

- nächste/vorherige Folie anzeigen
- Link öffnen
- Videodatei starten
- Audiodatei starten
- Bild vergrößern/verkleinern

mittels ihrer Tastenkombinationen umzusetzen. Nach einer Recherche bezüglichen der entsprechenden Tastenkombinationen wurde festgestellt, dass PowerPoint keine Tastenkombinationen für alle geplanten Funktionen bereitstellt.

Die Funktionen, „Nächste Folie“ und „Vorherige Folie“, lassen sich mit der Simulation der rechten und linken Pfeiltaste realisieren. Für die restlichen geplanten Funktionen gibt es keine definierten Tasten. Daher musste für diese Funktionen eine andere Lösung gefunden werden. Nach weiteren Recherchen und Tests mit PowerPoint wurde festgestellt, dass es zwar keine separaten Tastenkombinationen für diese Funktionen gibt, jedoch lassen sich Videos, Audios oder Links während einer Präsentation mittels dem Tabulator auswählen und über „ENTER“ ausführen.

Die Funktionen „Bild vergrößern“ und „Bild verkleinern“ lassen sich auch mit dieser Erkenntnis nicht realisieren, allerdings wurde erkannt, dass diese Funktionen auch nicht

zwingend bei PowerPoint Präsentation benötigt werden. Falls ein gezieltes Element hervorgehoben werden muss, ließe sich dies bei der Erstellung der Präsentation mittels vorhandener Effekte im PowerPoint umsetzen. Die Funktionen „Vergrößern“ und „Verkleinern“ sind eher bei anderen Präsentationstypen notwendig, z.B. PDF Präsentation.

Darüber hinaus ist eine weitere Herausforderung bei der Realisierung aufgetreten. Mit den Tasten „TAB“ und „ENTER“ lassen Links, z.B. zu einer Webseite, öffnen, allerdings befindet man sich anschließend außerhalb der Präsentationen. Daher ist eine weitere Funktion notwendig, um zur Präsentation zurückzukehren.

5.1.1 Schlussendliche Definition der Funktionen und Gesten

Die nachfolgende Tabelle zeigt alle Funktionen und Gesten, die für die Realisierung dieser Arbeit umgesetzt werden.

Funktion	Simulierte Taste	Geste
Nächste Folie	• Rechte Pfeiltaste 	• Wischgeste mit rechter Hand nach rechts
Vorherige Folie	• Linke Pfeiltaste 	• Wischgeste mit linker Hand nach links
Element (Video, Audio, Link) auswählen	• Tabulator (Tab - Taste) 	• offene linke Hand neben dem Körper
Element ausführen	• ENTER 	• offene rechte Hand neben dem Körper
Fenster schließen	• Alt + F4 	• geschlossene linke Hand neben dem Körper

Tabelle 5-1: Definition aller Funktion und Gesten

5.2 Erstellung eigener Gesten

Wie aus dem vorherigen Kapitel bekannt, wurde sich für die Umsetzung der Gesten für den „Visual Gesture Builder“ entschieden. Um Gesten mittels VGB zu erstellen, müssen diese zunächst in Form eines Videos vorhanden sein. Dafür empfiehlt sich die Verwendung des KINECT Studios. Mit diesem Programm kann man Daten von dem KINECT Sensor aufnehmen und abspielen. Grundsätzlich kann man aber auch die Videos für die Gesten mit einer anderen Technik aufnehmen. Der Vorteil des KINECT Studio ist jedoch, dass sich die Videos direkt in der benötigten „xef“ - Datei Format befinden. Sollten sich die Videos in einem anderen Format befinden, müssen diese mit dem „KSConverter“ in eine „xef“ - Datei umgewandelt werden.

Für eine gute Geste empfiehlt es sich, mehrere Videos zu einer Geste mit fünf bis zehn Wiederholungen dieser innerhalb eines Videos zu erfassen. So erzielt man später eine höhere Genauigkeit.

Der erste Schritt für eine eigene Geste war das Anlegen einer neue „Solution“ im „Visual Gesture Builder“. Diese enthält zum Schluss alle erstellten Gesten. In der erzeugten „Solution“ wurde anschließend ein neues Projekt erstellt. Dafür öffnete sich folgendes Fenster (Abb. 5-1).

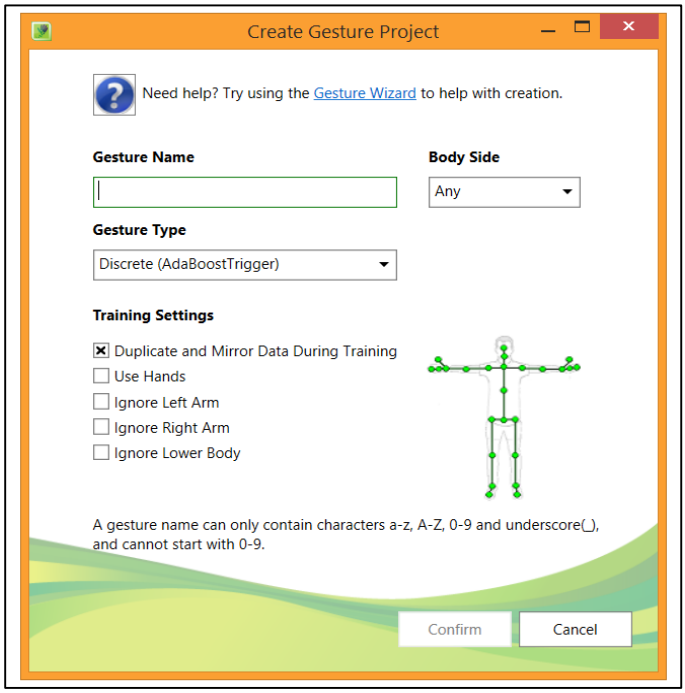


Abbildung 5-1: „Create Gesture Project“ - Fenster aus VGB

An diesen Punkt gab es zwei Möglichkeiten die gewünschte Geste zu erstellen. Wenn man bereits genau weiß, wie seine Geste später funktionieren soll, das heißt welcher Gestentyp, welche Körperteile etc. benötigt werden, kann dies über dieses Fenster auswählen. Nach der Erstellung einiger Gesten hat man dies bezüglich genug Erfahrung und

wird in der Regel diese Variante nutzen. Für Neulinge bot sich die Verwendung des Assistenten, den sogenannten „Gesture Wizard“, an. Der „Gesture Wizard“ kann im oberen Teil des Fensters ausgewählt werden. Dieser führt mit gezielten Fragen zur Geste ebenfalls zum gewünschten Ergebnis.

Da für die spätere Verwendung mehrere Gesten benötigt werden und bei der Realisierung sowohl diskrete als auch kontinuierliche Gesten zum Einsatz kommen, wird in den nächsten zwei Punkten auf die Unterschiede bei der Erstellung von diskreten und kontinuierlichen Gesten eingegangen.

5.2.1 Diskrete Geste

Noch mal zur Erinnerung: Eine diskrete Geste ist eine permanent gleichbleibende Geste. Diese Form der Geste soll beispielsweise bei der Funktion von „Auswahl von Elementen“ (Videos, Links) während einer Präsentation zum Einsatz kommen.

Für die Erstellung einer solchen diskreten Geste musste im geöffneten Fenster „Create Gesture Project“ der Projektname (Gesture Name), der Gestentyp (Gesture Type) und die benutzte Körperseite (Body Site) bestimmt werden.

Anschließen stehen einem noch ein paar Einstellungsmöglichkeiten zur Spezifizierung der Geste, z.B. das Ignorieren der unteren Gelenkpunkte, zur Verfügung. Bei der Geste zum Auswählen von Elementen konnte auf die Gelenkpunkte des unteren Körpers sowie des rechten Armes verzichtet werden, da diese für Ausführung uninteressant sind. Hingegen ist die Benutzung der Hand sehr wichtig, weil die Geste nur bei offener Hand ausgeführt werden soll. Die Auswahl von „Duplicate and Mirror Data During Training“ war in diesem Fall auch irrelevant. Diese Auswahl kommt nur bei symmetrischen Gesten, z.B. beim Springen zum Einsatz, weil sich dabei die linke und rechte Seite nicht unterscheidet. Für die Erstellung dieser diskreten Geste wurde die Auswahl daher folgendermaßen getroffen (Abb. 5-2):

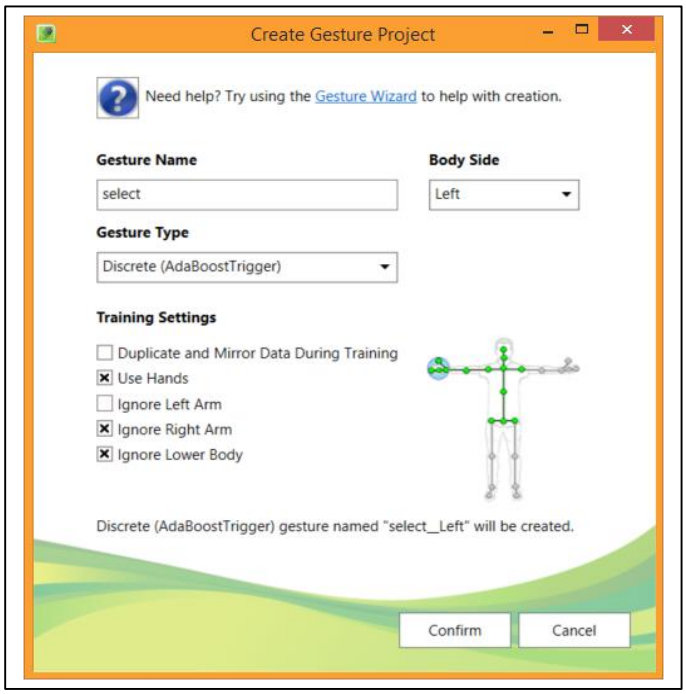


Abbildung 5-2: „Create Gesture Project“ - Fenster mit Einstellungen für diskrete Geste

Nach dem Bestätigen der Einstellung wurde das Projekt erstellt. Man erhielt in der Projektstruktur innerhalb der erstellten Solution zwei Einträge, „select_Left“ und „select_Left.a“.

„select_Left“ ist der „Trainingsbereich“. Hier wird mittels den erstellten Videos die Geste erfasst. „select_Left.a“ ist der „Analysenbereich“. Hier kann geprüft werden, wie gut oder schlecht die Geste funktioniert. Dazu lädt man ebenfalls Videos in das Programm. In diesen Videos wird anschließend geprüft, ob die erstellte Geste darin erkannt wird. Diese Form wird aber selten genutzt. Zur Überprüfung der Funktionalität der Gesten bietet sich die „Live Preview“ an. Diese zeigt ebenfalls ein visuelles Feedback über die Funktionalität der erstellten Geste (siehe nachfolgend im Punkt 5.2.3).

Die nächste Aufgabe war nun, die eigentliche Geste zu definieren (Abb. 5-3). Über das hinzugefügte Gestenvideo wurde im Steuerungsfenster der Abschnitt im Video mit der ausgeführten Geste markiert (getaggt). Da es sich bei dieser Form um eine diskrete Geste handelte, konnte diese nur zwei Zustände haben, „Wahr (True)“ oder „Falsch (False)“. Mittels Pfeiltasten und gehaltener SHIFT-Taste konnte der entsprechende Bereich markieren werden. Durch anschließendes Drücken der ENTER-Taste wurde der markierte Bereich auf True gesetzt. Das bedeutet, diese Auswahl entspricht der Geste. Durch Drücken der LEERTASTE konnte ein markierter Bereich auf False gesetzt werden. Dies ist aber nicht zwingend notwendig, da alles, was nicht auf True gesetzt wird, automatisch vom VGB als False erkannt wird. Daher sollte man beim Taggen der Geste beachten, dass eine neutrale Position des Körpers nicht mit zur Geste gehört.

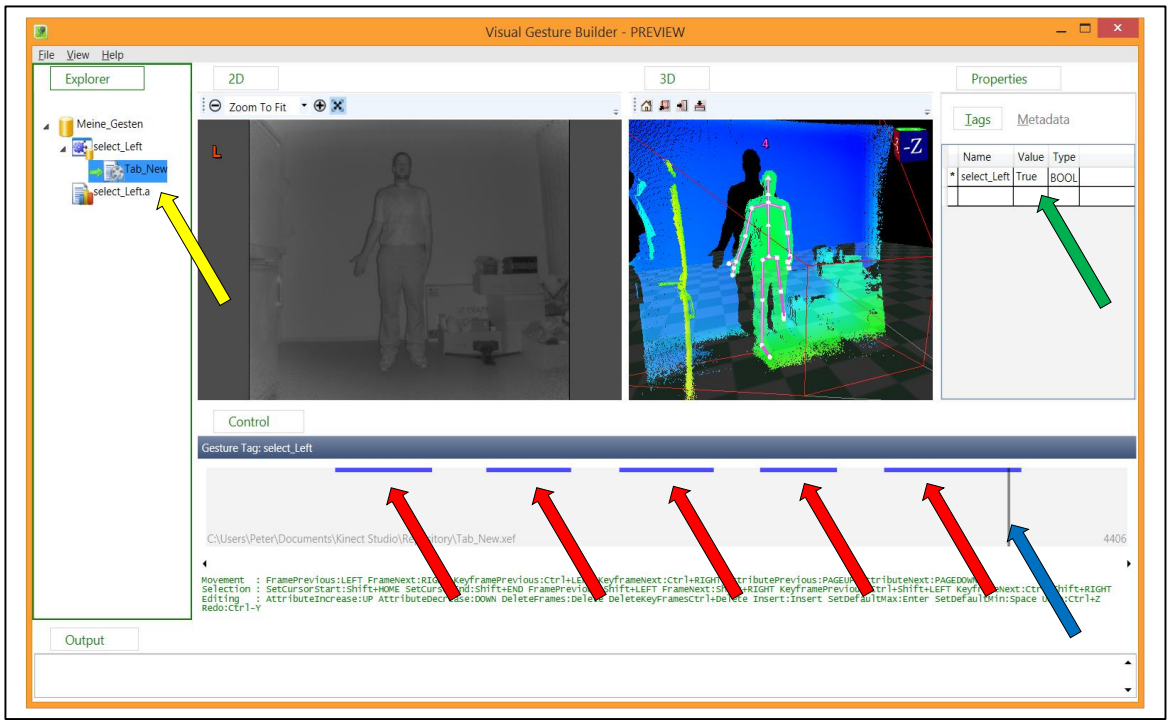


Abbildung 5-3: Ergebnis beim Definieren einer diskreten Geste im VGB

- Gelber Pfeil: zeigt den Clip, der für die Erfassung der Geste verwendet wird. Aktuell ist nur ein Clip hochgeladen, daher gibt es keinen weiteren zur Auswahl
- Blauer Pfeil: zeigt die Position des Wiedergabezeigers
- Rote Pfeile: zeigen die getaggten Gesten im Video
- Grüner Pfeil: zeigt den Zustand (in diesen Fall True)

5.2.2 Kontinuierliche Geste

Die Funktionen „Nächste Folie“ und „Vorherige Folie“ sollen mittels einer Wischgeste erfolgen. Bei dieser Form der Geste handelt es sich um kontinuierliche Gesten. Anders als bei diskreten Gesten, die nur True oder False sein können, haben kontinuierliche Gesten mehrere Zustände. Das Ziel bei der Erstellung dieser Gesten war daher, den Anfangs- und Endpunkt zu definieren.

Zunächst war die Vorgehensweise identisch der von diskreten Gesten. Im „Create Gesture Projekt“-Fenster wurden zuerst wieder die Einstellungen der Geste festgelegt. Auch hier sollte man wieder vorher wissen, welche Körperteile für die Geste benötigt werden. Daraus resultierte für diese Geste folgende Auswahl (Abb. 5-4):

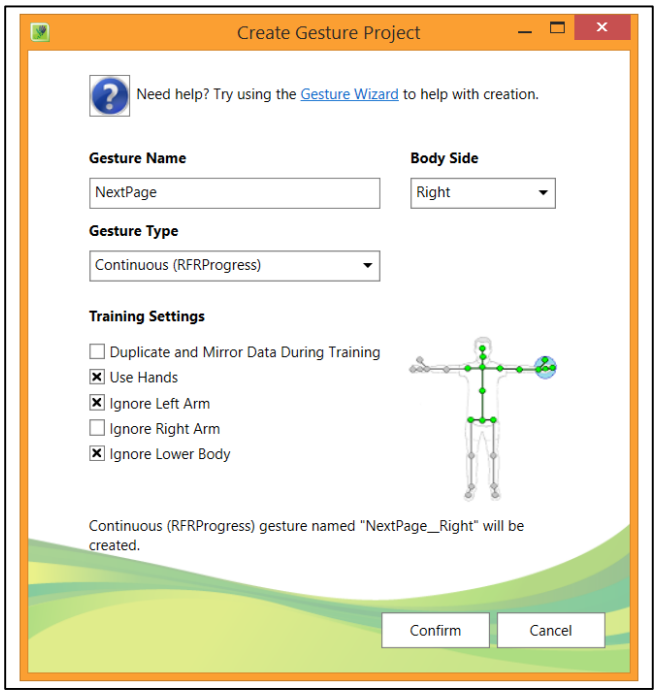


Abbildung 5-4: „Create Gesture Project“ - Fenster mit Einstellungen für kontinuierliche Geste

Wie bei der Erstellung der diskreten Geste wurden danach wieder zwei Einträge in die Projektstruktur hinzugefügt, „NextPage_Right“ und „NextPage_Right.a“.

Wo bei diskreten Gesten der gesamte Bereich der Geste mit True markierte wurde, muss bei kontinuierlichen Gesten ein Anfangspunkt mit 0 und ein Endpunkt mit 1 deklariert werden (Abb. 5-5). Der VGB ergänzt automatisch den Fortschritt der Geste mit Werten zwischen 0 und 1. Die Steuerung ist ähnlich der von diskreten Gesten. Mittels den Pfeiltasten, kann der Wiedergabezeiger bewegt werden. Über die LEERTASTE wurde der Anfangspunkt (Wert 0) gesetzt und mittels ENTER der Endpunkt (Wert 1).

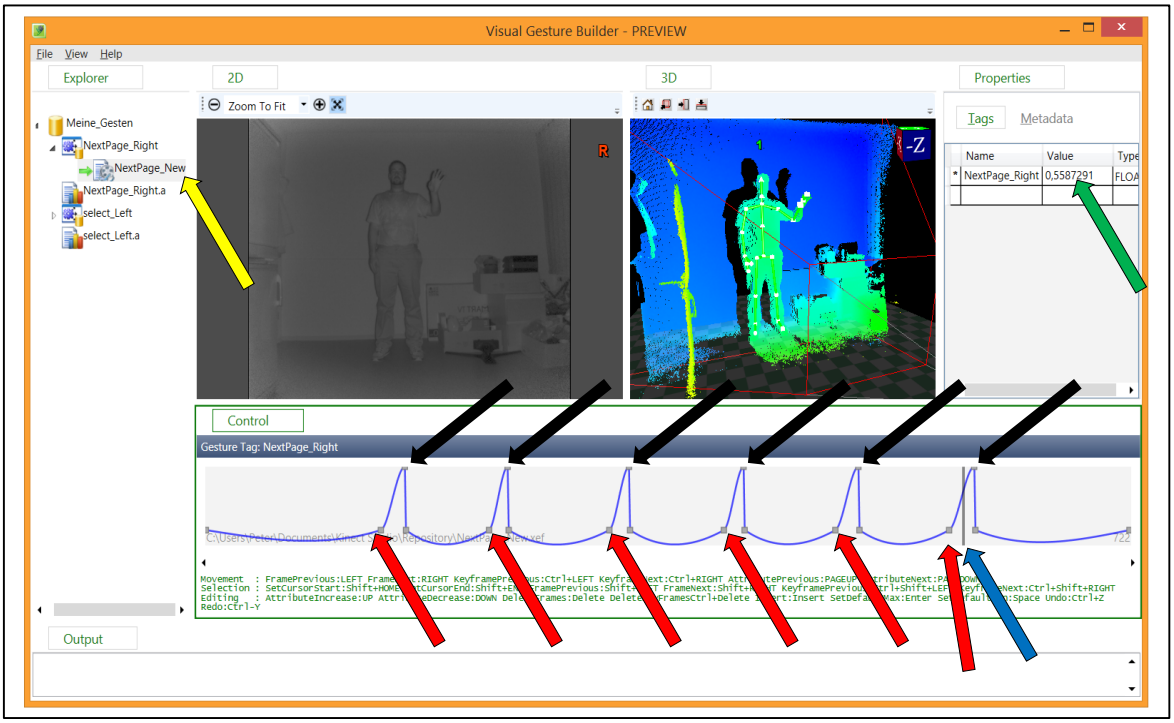


Abbildung 5-5: Ergebnis beim Definieren einer kontinuierlichen Geste im VGB

- Gelber Pfeil: zeigt den Clip der für die Erfassung der Geste verwendet wird. Aktuell ist nur ein Clip hochgeladen, daher gibt es keinen weiteren zur Auswahl.
- Blauer Pfeil: zeigt die Position des Wiedergabezeigers.
- Rote Pfeile: zeigen die Anfangspunkte der getaggten Gesten.
- Schwarze Pfeile: zeigen die Endpunkte der getaggten Gesten.
- Grüner Pfeil: zeigt den Wert des Fortschrittes der Geste an.

Bei der Erstellung von kontinuierlichen Gesten empfiehlt es sich, direkt nach einem Endpunkt (Wert 1) wieder ein 0 Wert zusetzen, damit in neutraler Position keine Geste erkannt wird. Abschließend kann die Geste wie auch diskrete Gesten mittels „Build“ erstellt werden.

5.2.3 Live Preview

Die „Live Preview“ ist ein integriertes Feature des „Visual Gesture Builder“, die dem Entwickler ermöglicht, schnell die Ergebnisse seiner erstellten Gestendatenbank in Echtzeit auf dem Computer zu sehen. Dazu muss die Datenbank in keine zusätzliche Anwendung integriert werden. Über einen Graphen von 0 bis 1 wird die Intensität der Geste dargestellt. Je höher der Graph desto besser wird die Geste erkannt.

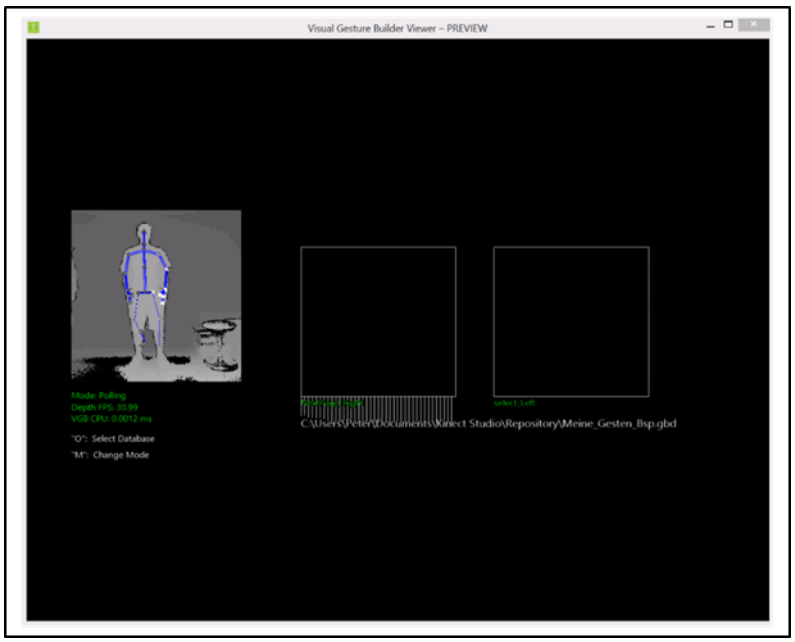


Abbildung 5-6: Live Preview

In der Abbildung (Abb. 5-6) kann man gut erkennen, dass in neutraler Position keine der beiden erstellten Gesten reagiert. Durch das Setzen eines 0 Wertes nach dem Endpunkt der kontinuierlichen Geste wird sogar aktuell ein negativer Wert erzielt.

Beim Ausführen der Wischgeste steigt der Graph der kontinuierlichen Geste an (Abb. 5-7). Da zu diesem Zeitpunkt nur ein Video für die Geste verwendet wurde, erreicht diese leider nicht komplett den Wert 1. Dies kann aber über weitere Videos noch verfeinert werden.

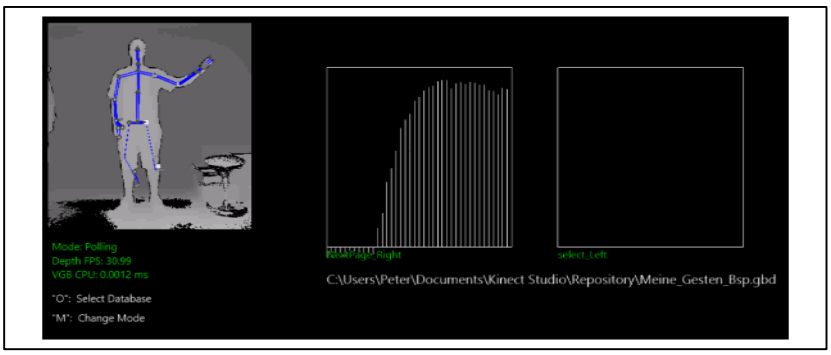


Abbildung 5-7: Live Preview beim Ausführen der kontinuierlichen Geste

Die Erkennung der diskreten Geste funktioniert dagegen bereits sehr gut. Die Geste wird voll erkannt und füllt den Graphen voll aus (Abb. 5-8).

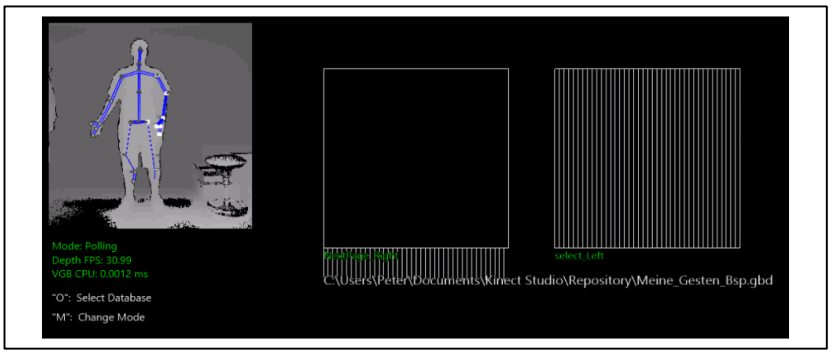


Abbildung 5-8: Live Preview beim Ausführen der diskreten Geste

Auch die Unterstützung der Handgelenkspunkte, welche beim Erfassen mittels „Use Hands“ hinzugefügt wurde, bewährt sich. So wird die Geste mit offener Hand voll erkannt, jedoch bei geschlossener Hand nicht. Nur leichte Ausbrüche des Graphen machen sich bei geschlossener Hand bemerkbar (Abb. 5-9).

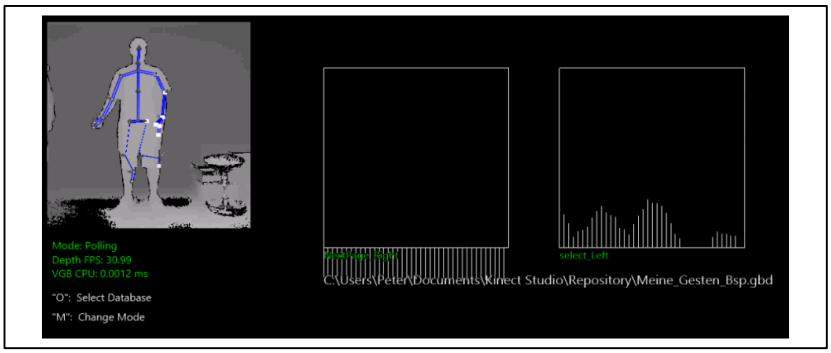


Abbildung 5.9: Live Preview beim Ausführen der diskreten Geste mit geschlossener Hand

5.3 Einbinden der Gesten

Nach dem im Punkt 5.2 beschriebenen Vorgehen wurden auch die restlichen Gesten erstellt. Nun musste die erstellte Gestendatenbank in die KINECT-Anwendung eingebunden werden. Dies war nicht sonderlich schwierig, allerdings müssen gerade anfangs ein paar Dinge beachtet werden, damit die erstellten Gesten funktionieren. Das Integrieren der Gestendatenbank erfolgte in die bereits erstellte KINECT-Anwendung der Skelettverfolgung, die das Resultat aus Punkt 4.2 war.

Für die Verwendung der Gesten musste zu Beginn der Verweis „Microsoft.Kinect.VisualGestureBuilder“ in die KINECT-Anwendung hinzugefügt sowie das Visual Gesture Builder „NuGet“-Paket installiert werden, außerdem musste noch die Gestendatenbank in das Projekt importiert werden (Abb. 5-10).

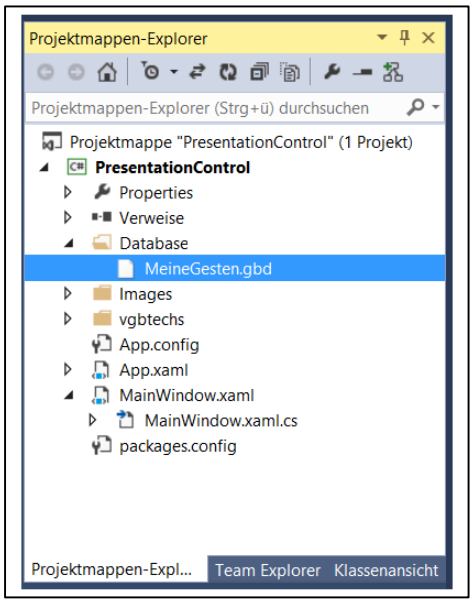


Abbildung 5-10: Projektmappe der KINECT - Anwendung

Wichtig, die Eigenschaften der „.gbd“-Datei müssen für die Verwendung angepasst werden. Dabei muss der Punkt „Buildvorgang“ auf „Inhalt“ und der Punkt „In Ausgabeverzeichnis kopieren“ auf „Kopieren, wenn neuer“ gesetzt werden (Abb. 5-11).

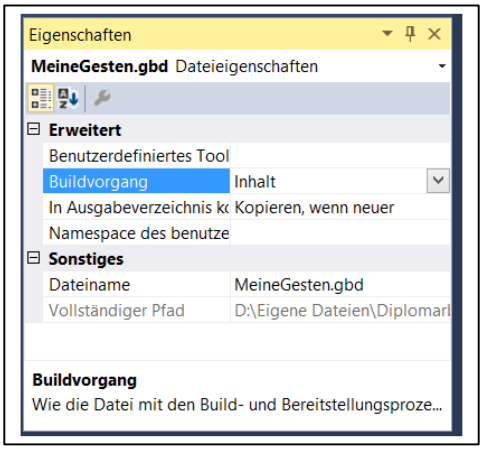


Abbildung 5-11: Eigenschaften der Gestendatenbank

Anschließend konnten die Gesten in den Quellcode eingebunden werden. Dazu wurden zunächst die Gestendatenbank und alle verwendeten Gesten aus der Datenbank als String deklariert. Hierbei muss sehr auf die korrekte Schreibweise geachtet werden. Der String muss exakt dem Namen der Geste in der Datenbank entsprechen, welchen man bei der Erstellung der Geste angegeben hat. Anderenfalls würde es später bei der Erkennung der Gesten, während der Ausführung der Anwendung, zum Fehler kommen, obwohl im Quellcode kein Fehler anzeigt wird.

Danach wurden noch der „VisualGestureBuilderFrameSource“ und „VisualGestureBuilderFrameReader deklariert und mit „Null“ initialisiert.

```
// Einbinden des Visual Gesture Builder---------------------------->//
// Datenbank
private readonly string gestureDatabase = @"Database\MeineGesten.gbd";

// Gesten
private readonly string nextPage = "NextPage_Right";
...

private VisualGestureBuilderFrameSource vgbFrameSource = null;
private VisualGestureBuilderFrameReader vgbFrameReader = null;
```

Quellcode 5-1: Deklaration der Gestendatenbank und Gesten

Anschließend erfolgte die Initialisierung dieser beiden Klassen im Konstruktor. Die VGB Frame Source erhielt als Übergabeparameter den KINECT Sensor und als Initiator die Tracking-ID Null. Die Initialisierung des VGB Frame Reader erfolgte, indem der Reader mittels „.OpenReader()“ geöffnet wurde. Wenn dieser anschließend ungleich Null ist, wird der Frame Reader mit dem „FrameArrived“-Event verknüpft.

Ebenfalls im Konstruktor resultierte noch das Laden der Gesten. Dies geschieht mittels „.AddGestures“ und „database.AvaiableGestures“.

```
this.vgbFrameSource = new VisualGestureBuilderFrameSource(kinectsensor, 0);
     this.vgbFrameReader = vgbFrameSource.OpenReader();
     if (vgbFrameReader != null)
     {
       this.vgbFrameReader.FrameArrived += this.Reader_GestureFrameArrived;
     }

using (VisualGestureBuilderDatabase database =
  new VisualGestureBuilderDatabase(this.gestureDatabase))
{
 try
 {
  vgbFrameSource.AddGestures(database.AvailableGestures);
 }
 catch (Exception e)
 {
 ...
```

Quellcode 5-2: Initialisierung des VGB-FrameReader und der VGB-FrameSource

In Folge darauf wurde das „FrameArrived“-Event „Reader_GestureFrameArrived“ erstellt. Hier musste zunächst über „AcquireFrame“ auf den aktuellen Frame zugegriffen werden. Nachdem geprüft wurde, ob der Frame ungleich Null und die Tracking - ID gültig ist, wird das „Dictionary“ (Wörterbuch) des Gestentyps geladen. Da in diesem Fall sowohl diskrete als auch kontinuierliche Gesten verwendet werden, müssen beide „Dictionaries“ geladen werden.

Je nach Gestentyp muss nun unterschieden werden, welche Gestentyp erkannt wurde. Beispielsweise für die Funktion „Nächste Folie“. Dabei handelt es sich um eine kontinuierliche Geste. Ist nun das Ergebnis der kontinuierlichen Geste (continuosResults) ungleich

Null, wird mittels einer „foreach“-Anweisung die Auflistung der Gesten in der „Frame-Source“ durchlaufen. Entspricht der Name der Geste den angegebenen String und dem entsprechenden Gestentyp, so wird die Geste erkannt. Damit die Funktion nicht bereits beim Heben der Hand ausgeführt wird, wird über eine weitere „If“-Anweisung geprüft, ob der Prozess der Geste größer als 0,9 ist.

```
if (continuosResults != null)
{
 foreach (Gesture gesture in this.vgbFrameSource.Gestures)
 {
  if (gesture.Name.Equals(this.nextPage) &&
       gesture.GestureType == GestureType.Continuous)
 {
   ContinuousGestureResult result = null;
   continuosResults.TryGetValue(gesture, out result);
   if (result != null)
   {
    if (result.Progress >= 0.9)
   {
     System.Diagnostics.Debug.WriteLine("Geste erkannt - Nächste Folie");
   }
  ...
```

Quellcode 5-3: Erkennung der kontinuierlichen Geste am Bsp. „Nächste Folie“

Das Erkennen der diskreten Gesten erfolgt ähnlich dem der kontinuierlichen Gesten. Zunächst muss das Ergebnis der diskreten Geste (discreteResults) ungleich Null sein. Die nachfolgenden Punkte sind identisch der kontinuierlichen Geste, natürlich entsprechend angepasst an diskrete Gesten. Zum Schluss wird jedoch kein Fortschritt des Prozesses überprüft, wann die Funktion ausgeführt werden soll, sondern eine Übereinstimmung. Diskrete Gesten haben bekanntlich nur zwei Zustände: „wahr“ oder „falsch“. Da bei kleinster Übereinstimmung eine diskrete Geste bereits wahr ist, würde die entsprechende Funktion ebenfalls sofort ausgeführt werden. Deshalb soll die Funktion erst ausgeführt werden, wenn die Geste erkannt (Detected) und eine Übereinstimmung von mehr als 0,9 hat.

```
if (discreteResults != null)
{
 foreach (Gesture gesture in this.vgbFrameSource.Gestures)
 {
  if (gesture.Name.Equals(this.select) &&
       gesture.GestureType == GestureType.Discrete)
 {
   DiscreteGestureResult result = null;
   discreteResults.TryGetValue(gesture, out result);
   if (result != null)
   {
    if (result.Detected && result.Confidence > 0.9)
    {
     System.Diagnostics.Debug.WriteLine("Geste erkannt - Element auswählen");
    }
  ...
```

Quellcode 5-4: Erkennung der diskreten Geste am Bsp. „Element auswählen“

Zum Schluss musste noch die „TrackingID“ gesetzt werden. Dies geschieht, indem in der „Reader_GestureFrameArrived“ der „VisualGestureBuilderFrameSource“, die „TrackingId“ des erkannten Körpers (Body) zugewiesen wird.

```
public void Reader_FrameArrived_BodyFrame(object sender,
       BodyFrameArrivedEventArgs e)
{
  ...
  if (body.IsTracked)
  {
   DrawClippedEdges(body, dc);
   //TrackingID für VGB
   vgbFrameSource.TrackingId = body.TrackingId;

   IReadOnlyDictionary<JointType, Joint> joints = body.Joints;
   Dictionary<JointType, Point> jointPoints =
              new Dictionary<JointType, Point>();
   ...
```

Quellcode 5-5: Setzen der Tracking ID im FrameArrived_BodyFrame

5.4 Umsetzung der Funktionalität

Nachdem nun alle Gesten in die KINECT-Anwendung integriert wurden und auch jede erkannt wird, war das nächste Ziel die Umsetzung der Funktionalität der Gesten. Dabei sollte jede Geste eine bestimmte Taste der Tastatur simulieren.

C# bietet zum Simulieren eines Tastendrucks mehrere Möglichkeiten an. In diesem Fall erfolgt die Realisierung mithilfe der „SendKeys“-Klasse [SEN_5-1].

Sobald die Geste erkannt wurde, wird die „SendKeys“-Klasse aufgerufen. Für die Umsetzung der Funktionalität wird auf die Methode „SendWait“ zurückgegriffen. Im Vergleich zur Methode „Send“ wartet diese die Verarbeitung der gesendeten Tastatureingabe in der Anwendung ab.

Das folgende Beispiel bezieht sich auf die Funktion „Nächste Folie“, dabei soll die rechte Pfeiltaste der Tastatur simuliert werden. Dies erfolgt durch die Angabe von „{RIGHT}“.

```
if (result.Progress >= 0.9)
{
  System.Diagnostics.Debug.WriteLine("Geste erkannt - Nächste Folie");
  SendKeys.SendWait("{RIGHT}");
}
...
```

Quellcode 5-6: Einbinden der SendKeys.SendWait - Methode

Grundsätzlich würde dies nun funktionieren, jedoch würde die „Send-Keys“-Methode so lange ausgeführt werden, wie die Geste erkannt wird. Dies hätte z.B. bei der Funktion „Nächste Folie“ die Folge, dass man nicht zur nächsten Folie gelangt, sondern mehrere

Folien überspringt. Aus diesem Grund wird der Prozess (Thread) nach dem ersten Ausführen gestoppt. Mithilfe der „Thread.Sleep“-Methode wird der Prozess für eine Sekunde angehalten.

```
using System.Threading;

...
if (result.Progress >= 0.9)
{
  System.Diagnostics.Debug.WriteLine("Geste erkannt - Nächste Folie");
  SendKeys.SendWait("{RIGHT}");
  Thread.Sleep(1000);
}
```

Quellcode 5-7: Setzen der Thread.Sleep - Methode

5.5 Weiterentwicklung der KINECT - Anwendung

Prinzipiell ist die Funktionalität der prototypischen Lösung dieser Arbeit realisiert. Beim Aufruf des Programms kann eine PowerPoint-Präsentation mithilfe der erfassten Gesten gesteuert werden.

Da die KINECT - Anwendung in diesen Zustand aber kein weiteres visuelles Feedback als die Ausgabe der Skelettverfolgung gibt, wurde entschieden, eine angemessene grafische Oberfläche mit weiteren Funktionen zur besseren Handhabung zu implementieren.

Damit die Gestensteuerung nicht automatisch beim Aufruf des Programms gestartet wird, wurde ein „Start“ - und „Beenden“ - Button hinzugefügt. Über diese Buttons kann der Nutzer selbst entscheiden, wann die Gestensteuerung aktiviert werden soll.

Neben Ausgabe der Skelettverfolgung verfügt die KINECT-Anwendung nun zusätzlich über die Ausgabe der Farbkamera und der Körperkontur. Beim Aufruf des Programms wird zunächst das Farbkamerabild der KINECT ausgegeben. Über Radiobuttons kann der Nutzer zwischen dem Farbkamerabild und dem Bild der Körperkontur wählen. Sobald jedoch die Gestensteuerung über den „Start“-Button aktiviert wird, wechselt die Ausgabe automatisch auf die Skelettdarstellung.

Um nicht den Überblick zu verlieren, in welchem Zustand sich das Programm befindet, wurde zusätzlich eine Statusleiste implementiert. Ist die KINECT am PC angeschlossen und vom Programm erkannt, wird dem Nutzer nach Aufruf des Programms in der Statusleiste „KINECT ist bereit“ angezeigt. Falls der Sensor nicht betriebsbereit ist, weil beispielsweise vergessen wurde den KINECT Adapter am Strom anzuschließen, würde dem Nutzer der Status „KINECT nicht gefunden“ angezeigt werden. Die KINECT-Anwendung wurde so programmiert, dass sich die Gestensteuerung nur im Zustand „KINECT ist bereit“ aktivieren lässt. Anderenfalls erhält man beim Startversuch eine Fehlermeldung (Abb. 5-12).

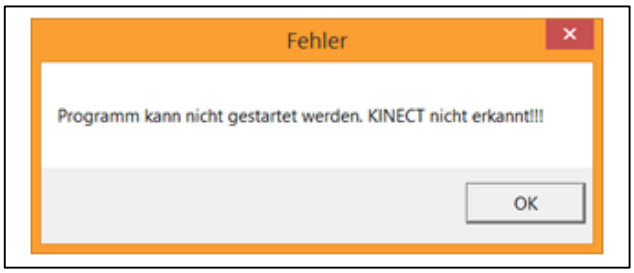


Abbildung 5-12: Fehlermeldung, wenn KINECT nicht erkannt wurde

Konnte die Gestensteuerung erfolgreich gestartet werden, wechselt die Statusleiste in den Zustand „PowerPoint Steuerung gestartet“ und der Nutzer erkennt sofort, dass sich das Programm im aktivierten Zustand befindet.

Da es passieren kann, dass man während einer Präsentation aus dem Sichtfeld der KINECT gerät, wurde dies ebenfalls berücksichtigt. Sollte der Redner das Sichtfeld der KINECT verlassen, sodass eine Gestenerkennung nicht mehr möglich ist, wird ihm dies über ein Bild auf dem Bildschirm mitgeteilt (Abb. 5-13). Dieses Bild wurde so programmiert, dass es immer im Vordergrund angezeigt wird und automatisch nach zwei Sekunden wieder verschwindet, wenn der Redner sich wieder im Sichtfeld befindet.

Abbildung 5-13: Warnmeldung während der Präsentation

Außerdem dient die Benutzeroberfläche zur Darstellung der Steuerung. Wird die Gestensteuerung aktiviert, wird dem Nutzer die Steuerung angezeigt und er erhält einen Überblick über die Funktionen.

Nach diesen Anpassungen ist das Resultat der KINECT-Anwendung folgendermaßen:

- Programm gestartet:

Abbildung 5-14: KINECT - Anwendung nach Aufruf des Programms

- Gestenerkennung aktiviert:

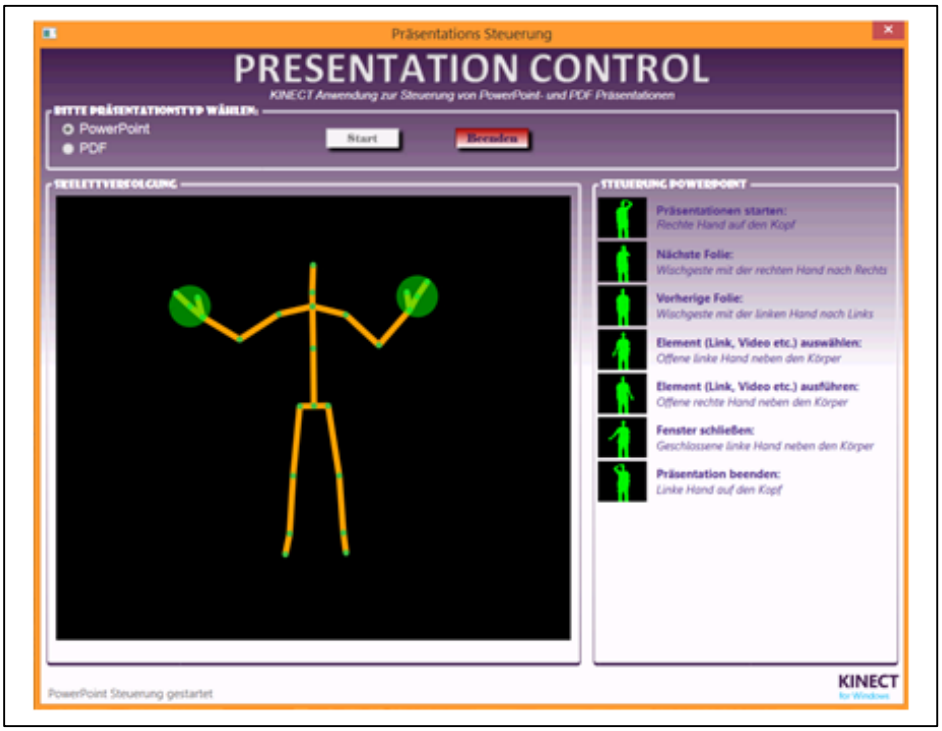


Abbildung 5-15: KINECT - Anwendung nach Aktivierung der Gestensteuerung

6 Validierung

Dieses Kapitel beschäftigt sich mit der Überprüfung der Funktionalität der prototypischen Lösung. Dabei wird zunächst die Vorgehensweise zur Überprüfung der Funktionalität betrachtet. Anschließend wird erklärt, was bei der Wahl der Testpersonen beachtet wurde und wie die Durchführung der Tests erfolgte.

6.1 Vorgehensweise

Für die Validierung sollen mehrere Tests mit der entwickelten prototypischen Lösung durchgeführt werden. Anhand dieser Tests wird anschließend überprüft, wie gut die Gestenerkennung der KINECT v2 funktioniert.

Für diese Tests wurde eine spezielle PowerPoint-Präsentation angefertigt, die alle realisierten Funktionen enthält. Die PowerPoint-Präsentation spiegelt sich so wider, dass der Testperson in der Präsentation Aufgaben gestellt werden, die die Testperson ausführen muss. Insgesamt enthält die Präsentation sechs Aufgaben, von „Gehe zur nächsten Folie“ bis, bei mehreren Links zur Auswahl, einen gezielten Link zu öffnen.

Abbildung 6-1: Auszug aus der Test PowerPoint Präsentation

Nach Abschluss der PowerPoint-Präsentation sollten die Testpersonen ihre Eindrücke zur Steuerung in einem angefertigten Feedbackbogen wiedergeben. Neben allgemeinen Fragen zur seiner Person, die die Unterscheidung zwischen den Testpersonen darstellen (Einzelheiten zu den Kriterien der Testpersonen im nächsten Punkt dieses Kapitels) und seinen Erfahrungen im Umgang mit KINECT und Präsentationen, wird gezielt nach der Funktionalität und Bedienbarkeit jeder einzelnen Geste gefragt.

Das Ziel soll schlussendlich sein, herauszufinden, inwieweit das Konzept der Steuerung von PowerPoint-Präsentationen mittels Gestensteuerung sinnvoll und durch die KINECT realisierbar sind. Ob auftretende Fehler an der Erfassung der KINECT bei bestimmten Personengruppen oder generell an der Massenkompatibilität der erstellten Gesten liegen.

6.2 Testpersonen

Um ein möglich großes Spektrum an Personen abzudecken, wurde bei der Auswahl der Testpersonen gezielt darauf geachtet, dass diese sich möglichst klar unterscheiden. Die zwei wichtigsten Kriterien diesbezüglich waren die Größe und der Körperbau. Da die Gesten nur mit Videos des Autors dieser Arbeit erstellt wurden, sollte so herausgefunden werden, ob die KINECT automatisch in der Lage ist, die Erkennung der Gesten an Person mit unterschiedlicher Größe und Körperbau, anzupassen. Sollte es zu Problemen bei der Gestenerkennung, bei einer bestimmten Personengruppe (große, kleine, korpulente oder dünne Personen), kommen, so kristallisiert sich diese dadurch heraus.

Zur Unterteilung dienen die drei bekannten Körperbautypen:

- Ektomorph - Neigung zu Schlankheit
  Sind dünne Körper mit schmächtigem Körperbau. Es sind Leichtgewichte und sie haben kleine Gelenke.

- Mesomorph - Neigung zu Muskulosität
  Sind im Schnitt athletische Körper mit wenig Körperfett.

- Endomorph - Neigung zu Adipositas
  Sind Körper mit einer starken Knochenstruktur mit insgesamt größeren Körpermaßen und höherem Körperfettanteil.

Ein weiteres Kriterium bei der Wahl der Testperson war das Alter. Unterteilt in die Altersklassen:

- Unter 25-Jährige
- 25 bis 34-Jährige
- 35 bis 45-Jährige
- Über 45-Jährige

Es sollten Testpersonen aus jeder Altersklasse an der Validierung teilnehmen. Bei der Unterteilung in Altersklassen lag der Schwerpunkt nicht zwingend bei der Überprüfung der Funktionalität, sondern diente eher zur Überprüfung der Akzeptanz dieser neuen Steuerungsmethode.

Darüber hinaus wurde versucht, bei der Wahl der Testpersonen darauf zu achten, dass bereits Erfahrungen mit PowerPoint bzw. mit dem Wiedergeben von Präsentationen bestehen.

Dies konnte nicht immer berücksichtigt werden, jedoch gerade bei der Frage, ob die Gestensteuerung bei Präsentation sinnvoll ist, hatten die Antworten von den Testpersonen mit viel Erfahrung eine höhere Bedeutung.

6.3 Testdurchführung

Die Tests fanden an unterschiedlichen Tagen sowie an verschieden Orten statt. Testpersonen aus allen genannten Kriterien haben an den Tests teilgenommen.

Die Tests erfolgten ohne große Einweisung in die Steuerung. Die Testpersonen erhielten eine PDF (Anlagen) mit der Aufgabenstellung, einer Übersicht über die Steuerung und den anschließend auszufüllenden Feedbackbogen. Die KINECT und die Software wurden für die Tests bereits gestartet. Das bedeutet, die KINECT-Anwendung wurde für die Testpersonen aufgerufen und die Gestenerkennung aktiviert. Außerdem wurde bereits die PowerPoint-Präsentation geöffnet.

Die Testpersonen stellten sich individuell vor die KINECT und versuchten die Präsentation ohne weitere Hilfen zu absolvieren, anschließend gaben sie ihre persönlichen Eindrücke im Feedbackbogen wieder.

7 Ergebnisse und Ausblick

In diesem abschließenden Kapitel wird das Ergebnis dieser Arbeit zusammengefasst und eine Bewertung der gewonnenen Erkenntnisse aus Sicht des Autors vorgenommen. Zum Schluss zeigt ein Ausblick Weiterentwicklungspotenziale und die Verwendung in der Praxis auf.

7.1 Ergebnis

Die unter dem Namen „PresentationControl“ entwickelte KINECT-Anwendung stellt ein Verfahren bereit, dass mittels definierter Gesten eine Steuerung von PowerPoint Präsentationen ermöglicht. Dafür wurden die Funktionen, die für eine professionelle Präsentation notwendig sind, spezifiziert und anschließend mithilfe der von „Visual Gesture Builder“ erstellten Gesten realisiert. Die Umsetzung der Anwendung erfolgte in C# und nutzt zur Erkennung die KINECT v2.

Die resultierende KINECT - Anwendung wurde anschließend über durchgeführte Tests auf ihre Robustheit und Massentauglichkeit überprüft. Während der Tests kam es dabei hin und wieder zu Problemen mit einzelnen Gesten. Jedoch konnte kein Problem bei einer gezielten Geste festgestellt werden. Wenn es zu Schwierigkeiten bei der Gestenerkennung kam, war es je nach Testperson eine andere Geste.

Es kristallisierte sich auch keine spezielle Personengruppe heraus, wo die Gestenerkennung immer am schlechtesten funktionierte. Sowohl bei Testpersonen mit beispielsweise einem ektomorphen Körperbautyp als auch bei Personen mit mesomorphen Körperbautyp funktionierte die eine oder andere Geste besser bzw. schlechter (Abb. 7-1).

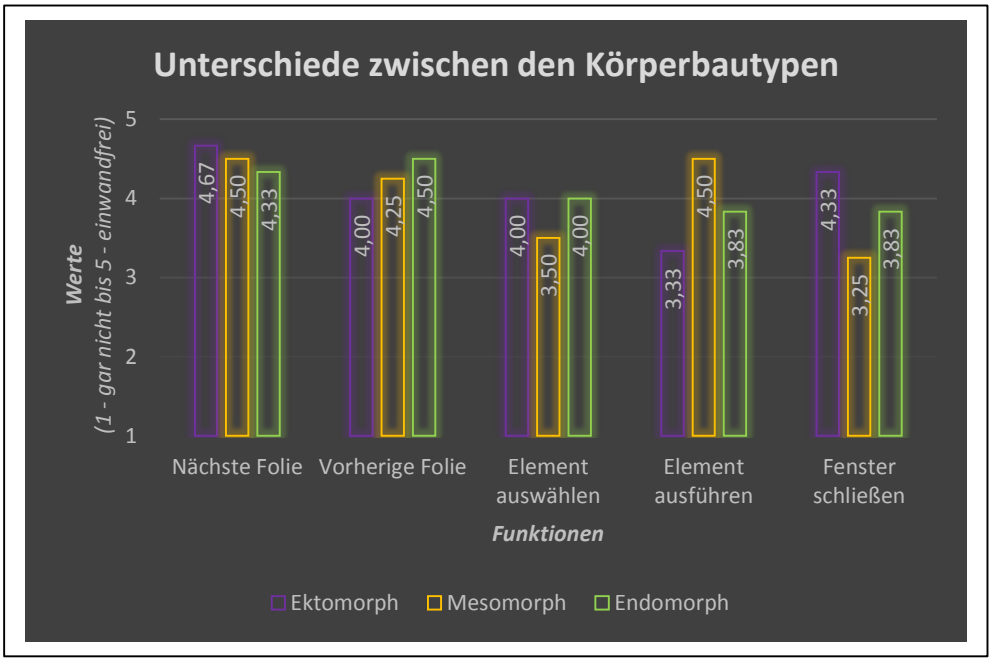


Abbildung 7-1: Übersicht über die Funktionalität der einzelnen Gesten

Die komplette Funktionalität der prototypischen Lösung wurde dennoch von den Testpersonen überwiegend mit „Gut“ bis „Sehr Gut“ bewertet (Abb. 7-2).

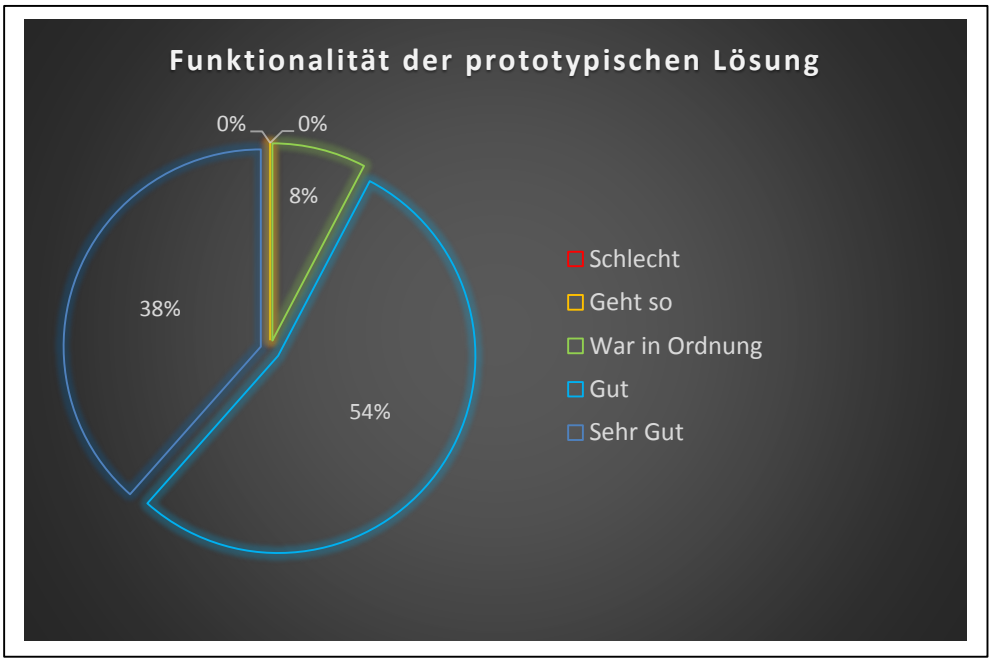


Abbildung 7-2: Einschätzung über die Funktionalität der prototypischen Lösung

Auch die Logik der einzelnen Gesten kam bei den Testpersonen gut an (Abb. 7-3). So erhielten die Wischgesten für die Funktionen „Nächste -“ und „Vorherige Folie“ ein perfektes Ergebnis. Auch die Gesten für die Funktionen „Auswählen -“ und „Ausführen von Elementen“ wurden mehrheitlich als logisch empfunden. Nur die Geste für die Funktion „Fenster schließen“, welche dazu dient, zur Präsentation zurückzukehren, konnte nicht voll überzeugen. Einige Testpersonen empfanden diese als nicht optimal.

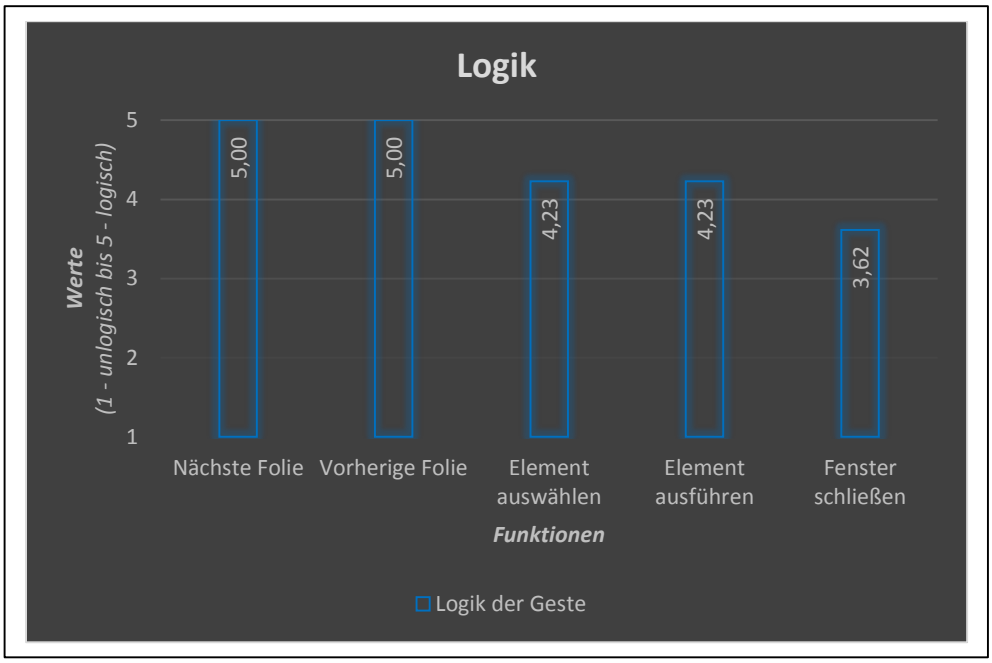


Abbildung 7-3: Übersicht über die Logik der einzelnen Gesten

Die Ergebnisse der Akzeptanz gegenüber der neuen Steuerungsmethode ergaben hingegen eine klare Tendenz (Abb. 7-4). Bei den Altersklassen der unter 25 - Jährigen und 25 bis 34 - Jährigen kam die Idee der gestenbasierten Steuerung von Präsentationen sehr gut an. Diese können sich auch vorstellen zukünftig mit dieser Technik Präsentationen zu halten. Sie gaben unter anderem als Grund für ihre Entscheidung an, dass diese Idee innovativ und zeitgemäß ist. Außerdem fanden sie es gut, dass keine Geräte mehr während der Präsentation gebraucht werden.

Die Altersklasse der 35 bis 45-Jährigen war dagegen geteilter Meinung. Einige der Testpersonen dieser Altersklasse waren der gestenbasierten Steuerung ebenso nicht abgeneigt wie die jüngeren Testpersonen und können sich durchaus zukünftig eine Verwendung vorstellen. Andere Personen dieser Altersklasse waren eher skeptisch und können sich nicht unbedingt eine Verwendung vorstellen.

Die Altersklasse der über 45-Jährigen war sich diesbezüglich allerdings relativ einig. Sowohl die Idee finden sie mehrheitlich unnötig und auch, trotz eines positiven Feedbacks bei Funktionalität, können sie sich eine Verwendung zukünftig nicht vorstellen.

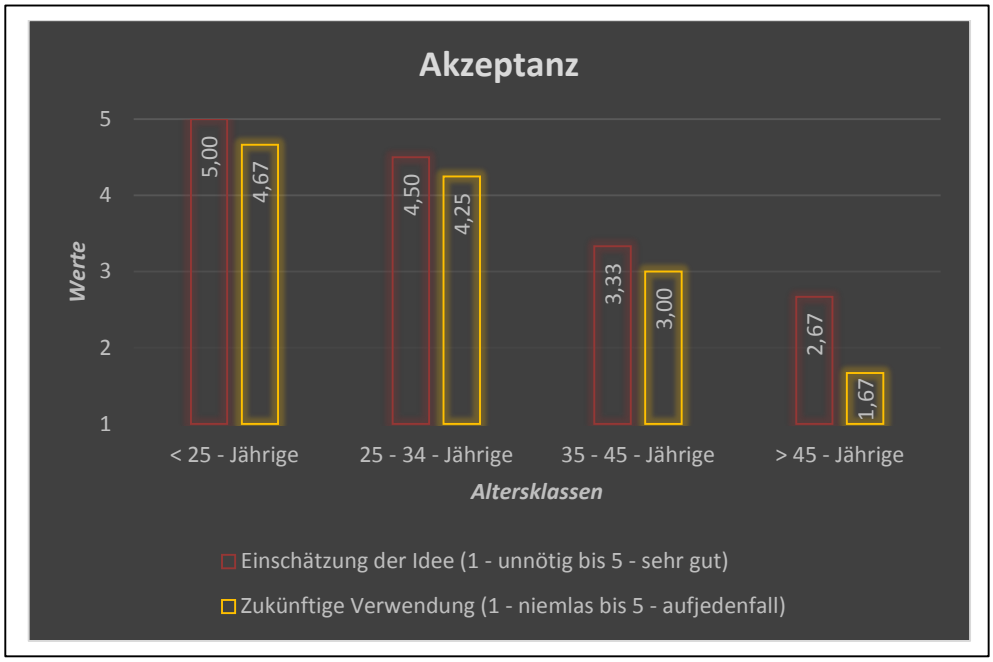


Abbildung 7-4: Übersicht über die Meinungen zur gestenbasierten Steuerung

Die Fragen über unnötige und weitere Funktionen ergaben, dass keine realisierte Funktion als unnötig empfunden wurde. Alle Funktionen werden für die Durchführung einer Präsentation benötigt. Einige Testpersonen würden sich dennoch zusätzlich noch eine Funktion zum „Vergrößern/Verkleinern“ wünschen.

Die abschließende Frage „Anmerkungen oder Verbesserungsvorschläge“ im Feedbackbogen wurde genutzt, um ein paar interessante Themen anzusprechen. So wurde z.B. das Thema „Steuerung außerhalb der Präsentation“ angesprochen. Auch das Verwenden von Kopfgesten wurde vorgeschlagen. So können sich Testpersonen vorstellen, dass ein Kopfnicken das Ausführen von Elementen aktiviert.

7.2 Erkenntnisse

Die im Rahmen dieser Diplomarbeit erstellte prototypische Lösung konnte mit seiner Funktionalität bei den Testpersonen überzeugen. Neben den Erkenntnissen über die Funktionalität, konnten auch allgemeine Erkenntnisse über die gestenbasierte Steuerung von Präsentationen mit KINECT, aus den Tests, gewonnen werden.

So ergab sich zunächst, dass der Mensch gut funktionierenden Gewohnheiten treu bleibt. Die Testpersonen der über 45 - Jährigen haben die letzten Jahre die Maus des Computers als Steuerungsmethode für Präsentation verwendet und sind damit gut zurechtgekommen, daher werden sie, obwohl sie die Idee der gestenbasierten Steuerung nicht komplett unnötig finden, auch zukünftig auf ihre gewohnte Steuerungsmethode zurückgreifen.

Des Weiteren ergab sich, dass die KINECT weiterhin nahezu unbekannt ist (Abb. 7-5). Über ein Drittel der Testpersonen konnte mit dem Begriff „KINECT“ nichts anfangen. Dies betraf sowohl Personen der unter 25 - Jährigen als auch der über 45 -Jährigen Testpersonen. Fast ein weiteres Drittel wusste zwar, was KINECT ist, hat diese aber noch nie genutzt. Auch wenn sich nicht alle Testperson für Technik begeistern, sondern diese eher nur zum Zwecke ihrer Arbeit verwenden, ist, auf Grundlage der mittlerweile fünfjährigen Marktpräsenz der KINECT und ihrer aufbietenden Möglichkeiten, das Ergebnis allerdings ziemlich enttäuschend.

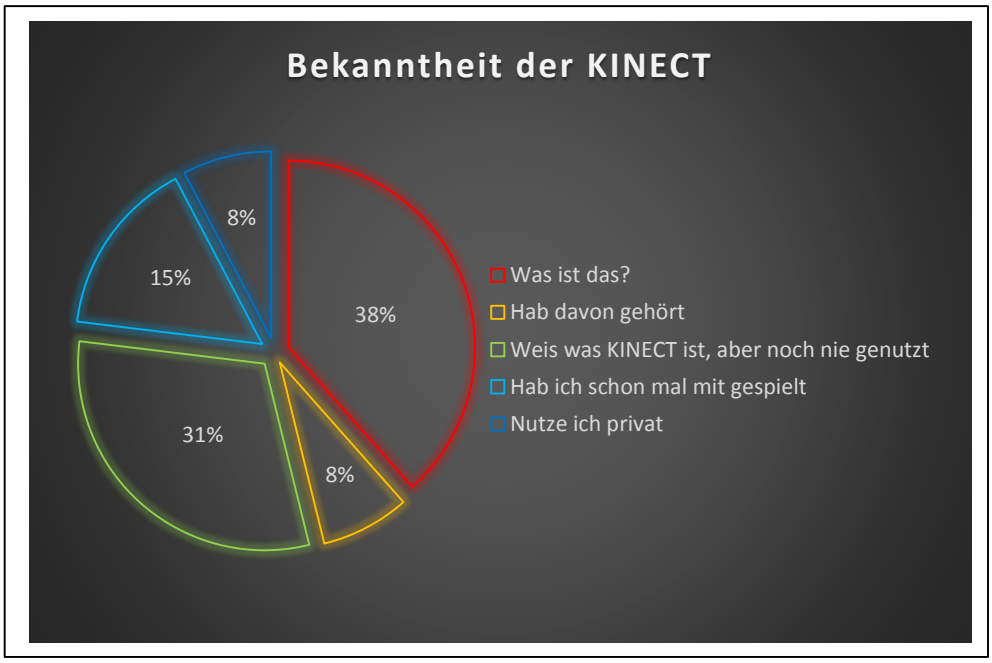


Abbildung 7-5: Übersicht über die Bekanntheit der KINECT

Nichtsdestotrotz ist die KINECT für die Realisierung einer gestenbasierten Steuerung von PowerPoint-Präsentationen absolut geeignet. Das Erkennen der erstellten Gesten ist für die KINECT kein Problem. Die Tests haben gezeigt, dass die Steuerung einer Präsentation mittels Gesten mithilfe der KINECT möglich ist. Auch die Entscheidung, die Realisierung der Gesten mittels dem „Visual Gesture Builder“ durchzuführen, war die richtige. Die Erstellung der Gesten war nach einer kurzen Eingewöhnung in das Tool relativ simpel und auch das Einbinden in die KINECT - Anwendung benötigte keine umfangreichen Kenntnisse.

Da es zur Verwendung des „Visual Gesture Builder“ nicht viele Resonanzen gibt, bestand bis zur Durchführung der Tests dennoch immer ein Risiko, über die Kompatibilität der Gesten zu anderen Personen. Es war während der Realisierung nicht klar, ob die erstellten Gesten, die nur auf Basis einer Person erstellt wurden, auch durch anderen Personen genutzt werden können. Die Tests haben aber gezeigt, dass die Kompatibilität der Gesten gegeben ist. Es bestehen keine Probleme bei der Erkennung unterschiedlicher Personen, jedoch ist eine Eingewöhnung in die Steuerung notwendig. Auftretende Probleme während der Tests resultierten nicht aus Fehlern bei der Gestenerfassung der KINECT oder

aus den mit dem VGB erstellten Gesten, sondern aus der Interpretation der Gesten. Obwohl die Gesten abgebildet und beschrieben wurden, kam es durch die Testpersonen zu den unterschiedlichsten Ausführungen der Gesten. Allerdings muss die Geste, damit die KINECT diese erkennt, fast exakt der erstellten Geste entsprechen, ansonsten wird diese nicht beachtet. Als den Testpersonen nach den Tests die korrekte Geste gezeigt wurde, konnte auch die für ihr Empfinden „nicht funktionierende“ Geste ausgeführt werden. Hätte man den Testpersonen vor der Durchführung der Tests eine persönliche Einweisung über die Ausführung der Gesten gegeben, statt nur über eine PDF, wäre wahrscheinlich das Ergebnis einzelner Gesten noch besser ausgefallen.

Zusammenfassend lässt sich sagen, dass eine gestenbasierte Steuerung von Präsentationen durchaus bereits möglich ist. KINECT bietet allemal die Möglichkeit dazu, ist jedoch für diese Anforderung unterfordert. KINECT wurde für wesentlich komplexere Anwendungen entwickelt.

Trotz großem Zuspruch für diese Steuerungsmethode gerade bei Personen bis 34 Jahre, ist eine regelmäßige Verwendung in nächster Zeit noch unwahrscheinlich. Natürlich hätten Präsentationen mit dieser Form der Steuerung eine ganz andere Wirkung auf die Zuhörer. Ein Redner der sich frei im Raum bewegt (im Rahmen des Sichtradius der KINECT) kann Zuhörer mehr mitreisen als ein Redner, der nur am Computer steht.

Auch wenn die gestenbasierte Steuerung mit KINECT diesen Vorteil mit sich bringt, gibt es auch einige gravierende Nachteile. So müsste bei Verwendung der KINECT Kamera diese beispielsweise in einen Präsentationsraum fest installiert sein. Für einen mobilen Einsatz ist sie eher ungeeignet. Auch wenn die KINECT v2 Kamera viel Toleranz, aufgrund des größeren Sichtfeldes, der geringeren Lichtempfindlichkeit und dem Spielraum von 0,8 bis 4,5 Metern Abstand zur Kamera, bei der Positionierung bietet, ist es mit hinstellen und anschließen nicht sofort getan. Eine Überprüfung der Funktionalität ist bei jeder neuen Position empfehlenswert.

Außerdem haben die Tests eindeutig gezeigt, dass Erfahrung im Umgang mit dieser Steuerung benötigt wird. Anderes als bei Maus und Tastatur ist eine fehlerfreie Bedienung auf Anhieb nicht möglich, sondern bedarf Übung.

Darüber hinaus sollte man bedenken, ob die Anschaffungskosten der KINECT im Verhältnis zum Nutzen steht. Personen die eins zwei Präsentationen im Jahr durchführen, für die ist die Investition eventuell nicht geeignet.

7.3 Ausblick

Für eine zukünftige und marktreife Verwendung der gestenbasierten Steuerung von Präsentationen gibt es in manchen Bereichen durchaus noch Erweiterungs- bzw. Verbesserungsbedarf.

Die in dieser Arbeit entwickelten Lösung beschäftigt sich nur mit der Steuerung innerhalb der Präsentation. Dabei wurde das Öffnen und Schließen von Links zu Webseiten bereits berücksichtigt, allerdings ist eine Steuerung auf der geöffneten Webseite derzeit nicht realisiert. Hier besteht durchaus Entwicklungspotenzial. Eventuell wäre eine Kombination aus der in dieser Arbeit verwendeten Methode und der „Hand Pointer Gestures“-Funktion (Kapitel 4.3.2), für die Steuerung außerhalb der Präsentation denkbar.

Ein weiterer Punkt ist die KINECT selbst. Mit den durchgeführten Tests dieser Arbeit hat die KINECT gezeigt, dass sie in der Lage ist, als Erkennungsgerät für eine gestenbasierte Steuerung von Präsentationen zu fungieren. Die Gestenerkennung der KINECT v2 funktioniert dafür einwandfrei, dennoch ist die KINECT nicht die Ideallösung für eine Gestensteuerung für Präsentationen.

Der KINECT Sensor ist relativ groß, wenn dieser nicht fest installiert ist, ist der Sensor bei Verwendung von Präsentation vergleichsweise unpraktisch. Dies ist aber keine Kritik an der KINECT. Die KINECT ist nicht für einen mobilen Einsatz und darüber hinaus für wesentlich komplexere Aufgaben konstruiert. Neben der Gestenerkennung bietet die KINECT noch weitere Funktionen, daher wird dieser Platz im Gerät auch weitestgehend benötigt.

Die Gestensteuerung gibt es schon seit ein paar Jahren, aber die erste Generation der KINECT hat diese erst richtig populär gemacht. Während sich mit der Gestensteuerung für die Thematik dieser Arbeit auseinandergesetzt wurde, ergaben sich bei den Recherchen zu diesem Thema interessante Alternativen zur KINECT, z.B. die Senz3D-Kamera von Creative. Diese Gesten erkennende Kamera ist im Vergleich zur KINECT weiteraus kompakter. Die Senz3D-Kamera ist jedoch nur auf die Gestenerkennung spezialisiert, weshalb auch ein kompakteres Gehäuse möglich ist.

Außerdem zeigten die Recherchen bereits Einsatzgebiete der berührungslosen Interaktion durch Gesten in Praxis auf. So wird diese beispielsweise im Pflege- und Krankenhausumfeld verwendet. Pflegepersonal kann mittels Gestensteuerung sowohl IT-Systeme, die zur Überwachung des Patienten dienen, als auch die Einrichtung im Pflegezimmer steuern. Durch die berührungslose Steuerung lassen sich so Übertragungen von gefährlichen Keimen vermeiden bzw. reduzieren, die immer noch ein großes Problem im Pflege- und Krankenhausumfeld darstellen.

Des Weiteren findet man Gestensteuerung mittlerweile auch schon bei TV-Geräten. Kleine Gestenkameras befinden sich dabei oberhalb des TV - Gerätes. Diese nehmen die

Handbewegungen des Nutzers war, wodurch Funktionen wie „Programm umschalten“ ausgeführt werden.

So lässt sich zum Schluss sagen, die gestenbasierte Steuerung findet in der Praxis immer mehr Verwendung und auch die gestenbasierte Steuerung von Präsentationen ist kein Traum mehr. Mithilfe von der Microsoft KINECT kann dies durchaus bereits genutzt werden. Damit sie sich dennoch in der Praxis neben den bevorzugten Steuerungsmethoden von Maus und Tastatur behaupten kann, benötigt es noch etwas Entwicklung hinsichtlich der Mobilität der Kameras und der generellen Handhabung während der Präsentation. Vermutlich wird ein richtiger Durchbruch erst erfolgen, wenn z.B. Laptops statt mit herkömmlichen Webcams mit Gesten erkennenden Kameras ausgestattet werden.

Glossar

.NET	Unter .NET versteht man eine von Microsoft entwickelte Software-Plattform zur Programmierung von Anwendungsprogrammen. Sie verfügt über eine große Anzahl an Klassenbibliotheken und bietet die Zusammenarbeit von verschiedenen Systemen oder Techniken. Dadurch ist .NET auf verschiedenen Plattformen verfügbar und unterstützt die Verwendung einer Vielzahl von Programmiersprachen.
API	API ist eine Schnittstelle für den Programmierer zur Anbindung an das System. Eine API besteht aus Funktionen, Konstanten und Variablen und stellt Befehle und Routinen, die von dem Betriebssystem oder einer Betriebssystemerweiterung kommen, als Programmierhilfen bereit.
Dictionary	In der Programmierung ist ein „Dictionary“ eine Auflistung, dass für das jeweilige Bereich Schlüssel und Werte bereitstellt.
DLL	DLL bezeichnet allgemein eine dynamische Programmbibliothek. Eine DLL enthält sowohl Code als auch Daten. Mithilfe von DLLs lässt sich ein Programm in separate Komponenten aufgliedern. Dies hat den Vorteil, dass ein Programm schneller geladen wird. Da das Laden der jeweiligen Komponente nur dann erfolgt, wenn die entsprechende Funktion benötigt wird.
Framework	Im Software-Engineering ist ein Framework ein modernes Rahmenwerk, das dem Programmierer den Entwicklungsrahmen für seine Anwendungsprogrammierung zur Verfügung stellt und damit die Software-Architektur der Anwendungsprogramme bestimmt. Das Framework wird vorwiegend in der objektorientierten Programmierung eingesetzt und umfasst Bibliotheken und Komponenten wie Laufzeitumgebungen und stellt die Designgrundstruktur für die Entwicklung der Bausteine zur Verfügung.

Konstruktor	Als Konstruktoren werden in der Programmierung spezielle Prozeduren bzw. Methoden bezeichnet, die beim Erzeugen und Auflösen von Objekten und Variablen aufgerufen werden. Die Aufgabe von Konstruktoren ist, Objekte in einen definierten Anfangszustand zu bringen und so benötigte Ressourcen zu reservieren, insofern diese zum Zeitpunkt der Objekterstellung bereits bekannt sind.
NuGet - Paket	NuGet ist eine freie Open-Source-Paketverwaltung für das .NET Framework. Es wird als Visual-Studio-Erweiterung ausgeliefert. NuGet unterstützt auch native Pakete, welche in C++ verfasst wurden. Seit der Einführung 2010 hat sich NuGet in ein Ökosystem für Softwarewerkzeuge und Softwaredienste entwickelt.
Photonen	Als Photon wird eine Lichtportion bezeichnet, die durch eine elementare Anregung eines elektromagnetischen Feldes entsteht.
PowerPoint	Ist ein von Microsoft entwickeltes Computerprogramm, mit dem sich interaktive Präsentationen unter Windows und Mac OS erstellen lassen.
Präsentation	Ist das Darstellen einer Sache vor einem Publikum.
Quellcode	Auch Quelltext (englisch source code) genannt, ist in der Informatik der für Menschen lesbare, in einer Programmiersprache geschriebene Text eines Computerprogrammes. Er beschreibt exakt und vollständig ein Programm, dass automatisch von einem Computer in Maschinensprache übersetzt werden kann. Der Quellcode eines Programms kann aus mehreren Dateien, in unterschiedlich Form bestehen.
Shortcut	Shortcuts sind Tastenkombinationen. Mit einer Tastenkombination können bestimmte Steuerbefehle an ein Programm gesendet werden. Nahezu alle modernen Programme besitzen heutzutage Shortcuts um das Arbeiten schneller und effektiver zu gestalten. Zu den bekanntesten Shortcuts gehören die Funktionen „Kopieren“ und „Einfügen“, welche mit der Tastenkombination „STRG“ + „C“ für kopieren und „STRG“ + „V“ für einfügen ausgeführt werden können.

Softwarearchitektur	In der Informatik beschreibt eine Architektur die grundlegenden Komponenten und deren Zusammenspiel innerhalb eines Softwaresystems.
WPF	WPF ist ein Grafik-Framework und Teil des .NET Frameworks von Microsoft. Es handelt sich um eine neu eingeführte Bibliothek von Klassen, die zur Gestaltung von Oberflächen und zur Integration von Multimedia-Komponenten und Animationen dient. Sie vereint die Vorteile von DirectX, Windows Forms, Adobe Flash, HTML und CSS. WPF-Anwendungen können sowohl Desktop- als auch Web-Anwendungen sein.

Quellen

[WII_2-1] Nintendo Wii Verkaufszahlen in Deutschland
http://de.statista.com/statistik/daten/studie/30810/umfrage/absatz-von-spielekonsolen-in-deutschland-nach-plattformen/
verfügbar am 10.09.2015, 11.27 Uhr

[KIN_2-2] Schwächen und Grenzen KINECT 1. Generation
https://homepages.thm.de/~hg6458/semsts/Kinect%20als%20Eingabegeraet%20-%20Ausarbeitung_Georg.pdf
verfügbar am 13.09.2015, 15.47 Uhr

[KIN_2-3] Reichweite Tiefenkamera
https://go.microsoft.com/fwlink/p/?LinkID=403900
verfügbar am 22.09.2015, 13.26 Uhr

[KIN_2-4] Reichweite Skelettverfolgung
https://go.microsoft.com/fwlink/p/?LinkID=403900
verfügbar am 22.09.2015, 13.26 Uhr

[SDK_3-1] Definition SDK
http://www.itwissen.info/definition/lexikon/software-development-kit-SDK.html
verfügbar am 28.09.2015, 14.54 Uhr

[SDK_3-2] KINECT SDK v1.0
http://www.microsoft.com/en-us/download/details.aspx?id=28782
verfügbar am 29.09.2015, 09.20 Uhr

[SDK_3-3] KINECT Studio
https://msdn.microsoft.com/en-us/library/hh855389.aspx
verfügbar am 29.09.2015, 13.38 Uhr

[SDK_3-4] KINECT Fusion
https://msdn.microsoft.com/en-us/library/dn188670.aspx
verfügbar 29.09.2015, 13.04 Uhr

[SDK_3-5] Unterschiede und Neuerungen in den verschiedenen KINECT SDK Versionen
https://msdn.microsoft.com/en-us/library/jj663803.aspx#SDK_1_5
verfügbar am 29.09.2015, 13.44 Uhr

[SDK_3-6] KINECT SDK 2.0
https://msdn.microsoft.com/library/dn799271.aspx
verfügbar am 29.09.2015, 13.40 Uhr

[VGB_3-7] Visual Gesture Builder
https://msdn.microsoft.com/en-us/library/dn785304.aspx
verfügbar am 30.09.2015, 11.12 Uhr

[RFR_3-8] RFRProgress
https://msdn.microsoft.com/en-us/library/dn785524.aspx
verfügbar am 30.09.2015, 11.12 Uhr

[ADA_3-9] AdaBoostTrigger
https://msdn.microsoft.com/en-us/library/dn785522.aspx
verfügbar am 30.09.2015, 11.12 Uhr

[SDK_3-10] Angabe über unterstütze Programmiersprache
https://dev.windows.com/en-us/kinect/tools
verfügbar am 30.09.2015, 15.17 Uhr

[WIKI_3-11] Liste mit weiteren unterstützten Programmiersprachen von .NET
https://de.wikipedia.org/wiki/Liste_von_.NET-Sprachen
verfügbar am 01.10.2015, 10.02 Uhr

[OPEN_4-1] Onlineplattform OpenKinect
http://openkinect.org/wiki/Main_Page
verfügbar 21.10.2015, 10.09 Uhr

[FOR_4-2] „ForEach“ - Schleife zur Erfassung der aktiven KINECT
https://msdn.microsoft.com/en-us/library/hh973072.aspx
verfügbar 26.10.2015, 11.12 Uhr

[SDK_4-3] Höhere Auflösung ab SDK 1.6
https://msdn.microsoft.com/en-us/library/jj663803.aspx
#SDK_1pt6_M2
verfügbar 26.10.2015, 11.57 Uhr

[JOI_4-4] Auflistung aller Gelenkpunkte
https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx
verfügbar 27.10.2015, 14.13 Uhr

[SDK_4-5] Unterstützte APIs
https://msdn.microsoft.com/en-us/library/dn782041.aspx
verfügbar 29.10.2015, 13.37 Uhr

[SEN_5-1] SendKeys - Klasse
https://msdn.microsoft.com/de-de/library/System.Windows.Forms.SendKeys(v=vs.110).aspx
verfügbar 17.11.2015, 10.05 Uhr

Literatur

[HAN13] Hanna, Tam: Microsoft KINECT – Programmierung des Sensorssystems, dpunkt, Heidelberg, 2013 ISBN: 9783864900303

[JAN12] Jana, Abhijit: KINECT for Windows SDK Programming Guide, Packt Publishing Ltd., Birmingham, 2012 ISBN: 9781849692380

[THE14] Theis, Thomas: Einstieg in Visual C# 2013, Rheinwerk Computing, Bonn, 2014 ISBN: 9783836228145

[AAL15] Keiser, T.; Höß, O.; Klein, B.; Neuhüttler, J.; Schneider, H.; Vetter, T.: Das Projekt GeniAAL – Gestensteuerung im Pflegeumfeld, Books on Demand, Norderstedt, 2015 ISBN: 9783734769658

[GUN00] Gunnerson, Eric: C# - Die neue Sprache für Microsofts .NET-Plattform, Galileo Computing, 2000, ISBN: 3898421074

Anlagen

Anlage, Teil 1

- **PDF zur Durchführung der Validierung**

Validierung der prototypischen Lösung

Allgemeine Informationen

Beschreibung

Dieser Test wird im Rahmen meiner Diplomarbeit durchgeführt. Er dient zur Überprüfung der Funktionalität sowie der Genauigkeit der Gestenerfassung der entwickelten prototypischen Lösung zur Steuerung von PowerPoint Präsentationen mithilfe der KINECT v2.

Erwartungen und Zielsetzungen

Anhand diesem Test möchte ich überprüfen, wie gut die Gestenerkennung der KINECT v2 funktioniert. Ob auftretende Fehler an der Erfassung der KINECT oder an der Massenkompatibilität der erstellten Gesten liegt.

Das Ziel soll schlussendlich sein herauszufinden, in weit das Konzept der Steuerung von PowerPoint Präsentationen mittels Gestensteuerung sinnvoll und durch die KINECT realisierbar ist.

Testdurchführung

Verwendete Systeme

Zur Testdurchführung wird die von mir erstellte KINECT - Anwendung „PresentationControl“ und sowie eine explizit für diesen Test erstellte PowerPoint Präsentation verwendet. Um alle Funktion der KINECT Anwendung zu testen, wurden alle Elemente (Videos, Links etc.) in die PowerPoint - Präsentation integriert.

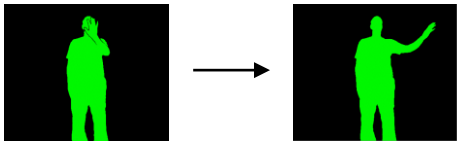
Aufgabenstellung

Die KINECT - Anwendung ist von mir für Durchführung des Tests gestartet. Bitte führe die PowerPoint Präsentation mit allen darin enthaltenen Aufgaben mittels der unten angegebenen Steuerung aus. Fülle bitte anschließend den beiliegenden Feedbackbogen nach Deinem persönlichen Empfinden aus.

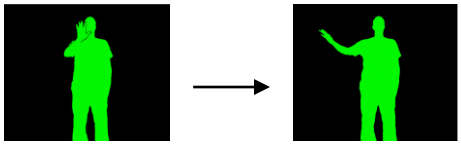
Steuerung

Präsentation starten | Rechte Hand auf den Kopf

Nächste Folie | Wischbewegung vor dem Kopf mit der rechten Hand nach rechts



Vorherige Folie | Wischbewegung vor dem Kopf mit der linken Hand nach links



Element (Video, Link, etc.) auswählen | Linker gesteckter Arm mit offener Hand neben dem Körper

Element (Video, Link, etc.) ausführen |Rechter gesteckter Arm mit offener Hand neben dem Körper

Fenster/Anwendung schließen | Linker gesteckter Arm mit geschlossener Hand abgespreizt neben dem Körper

Präsentation beenden | Linke Hand auf den Kopf

Vielen Dank für deine Unterstützung

Anlage, Teil 2

- **Feedbackbogen**

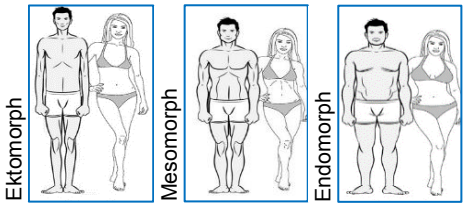
Feedbackbogen

Deine Meinung ist mir wichtig!!!
Nach Durchführung der Test PowerPoint - Präsentation möchte ich gern Deine persönliche Meinung dazu wissen. Bitte beantworte dazu die nachfolgenden Fragen. Dies nimmt noch etwa 5 - 7 Minuten Deiner Zeit in Anspruch.
Danke für Deine Unterstützung!

ALLGEMEIN	**1**	**2**	**3**	**4**	**5**

Größe in cm: ☐

Körperbautyp:



☐ ☐ ☐

Altersklasse:

unter 25-jährig ☐
25 bis 34-jährig ☐
35 bis 45-jährig ☐
über 45-jährig ☐

Kennen du KINECT?
(1 - Was ist das? bis 5 - Nutze ich privat)

Kennen du PowerPoint?
(1 - Was ist das? bis 5 - Nutze ich privat)

Wie oft hälst du Präsentationen?
(1 - Nie bis 5 - regelmäßig)

☐	☐	☐	☐	☐
☐	☐	☐	☐	☐
☐	☐	☐	☐	☐

Wie hast du bisher Präsentationen gesteuert?

Maus	Tastatur	Presenter
☐	☐	☐

andere Steuersmethode *(Bitte eintragen)* ☐

Einschätzung der Funktionalität der Gesten	1	2	3	4	5

NÄCHSTE FOLIE

Ist für dich die Geste für diese Funktion logisch?
(1 - unlogisch bis 5 - logisch)

Hat diese Geste auf deine Interaktion reagiert?
(1 - gar nicht bis 5 - einwandfrei)

Anmerkung oder Verbesserungsvorschlag:

VORHERIGE FOLIE

Ist für dich die Geste für diese Funktion logisch?
(1 - unlogisch bis 5 - logisch)

Hat diese Geste auf deine Interaktion reagiert?
(1 - gar nicht bis 5 - einwandfrei)

Anmerkung oder Verbesserungsvorschlag:

ELEMENT AUSWÄHLEN

Ist für dich die Geste für diese Funktion logisch?
(1 - unlogisch bis 5 - logisch)

Hat diese Geste auf deine Interaktion reagiert?
(1 - gar nicht bis 5 - einwandfrei)

Anmerkung oder Verbesserungsvorschlag:

ELEMENT AUSFÜHREN

Ist für dich die Geste für diese Funktion logisch?
(1 - unlogisch bis 5 - logisch)

Hat diese Geste auf deine Interaktion reagiert?
(1 - gar nicht bis 5 - einwandfrei)

Anmerkung oder Verbesserungsvorschlag:

FENSTER SCHLIEßEN

Ist für dich die Geste für diese Funktion logisch?
(1 - unlogisch bis 5 - logisch)

Hat diese Geste auf deine Interaktion reagiert?
(1 - gar nicht bis 5 - einwandfrei)

Anmerkung oder Verbesserungsvorschlag:

Einschätzung des Tools

	1	2	3	4	5
Wie ist die Funktionalität im ganzen betrachtet? *(1 - schlecht bis 5 - sehr gut)*					
Wie findest du die Idee generell Präsentationen mittels Gesten zu steuern? *(1 - unnötig bis 5 - sehr gut)*					
Kannst du dir Vorstellen zukünftig mittels KINECT Präsentationen durchzuführen? *(1 - niemlas bis 5 - aufjedenfall)*					

Warum hast du dich bei den letzten zwei Fragen so entschieden?

Findest du vorhandene Funktionen unnötigt?

Sollten weitere Funktionen vorhanden sein?

Sonstige Anregungen oder Kritik:

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Gera, den 02. Januar 2016

Peter Hofmann